# NASA Technical Memorandum 100636

NASA-TM-100636 19880018063

# FTMP DATA ACQUISITION ENVIRONMENT



Peter A. Padilla

July 1988



National Aeronautics and
Space Administration

**Langley Research Center**
Hampton, Virginia 23665

## SUMMARY

The Fault-Tolerant Multi-Processor (FTMP) test-bed data acquisition environment is described. The performance of two data acquisition devices available in the test environment are estimated and compared. These estimated data rates are used as measures of the data acquisition devices' capabilities.

The FTMP data acquisition environment consists of parallel data acquisition paths, with each path comprising several devices (with their associated software) which are connected in series to form a data path between the target system, FTMP, and the host system, a VAX minicomputer. Two parallel data acquisition paths, the Collins Test Adapter (CTA) and the 1553 data bus, comprise the original environment.

A new data acquisition path has been developed and added to the FTMP environment. This path increases the data rate available by approximately a factor of 8, to 379KW/s, while simplifying the experiment development process. The control software was developed and allows application programs written in the host VAX minicomputer to access the control software functions through the QIO system service interface.

Everything of interest in FTMP, e.g., system status and configuration data, appears on the bus periodically to be processed by the operating system or application software. The new acquisition system monitors FTMP's system bus and can trigger on any selected word that appears on the bus. Therefore, state and performance data, and information on fault effects can readily be obtained.

# INTRODUCTION

The FTMP data acquisition environment consists of parallel data acquisition paths, with each path comprising several devices with their associated software which are connected in series to form a data path between the target system, FTMP, and the host system, a VAX minicomputer. Two parallel data acquisition paths, the Collins Test Adapter (CTA) and the 1553 data bus, comprise the original environment.

Data acquisition rate limitations in both data acquisition paths motivated the design and implementation of a new data acquisition path to be added to the environment. The new data acquisition system (DAS) is designed to monitor, in real-time, the signals from FTMP's system bus. The system was interfaced to the redundant serial bus of the FTMP test-bed and allows a researcher to monitor and store the contents of FTMP's system bus during system operation. On this bus the state of the system is periodically transmitted to be processed by system software. The new acquisition path allows researchers to acquire and store this data efficiently.

The data acquisition system provides the following functions which will be explained in more detail in a later section :

1. continuously monitors a user-selected bus, searching for a user-specified trigger word;

2. starts acquisition (data storage in local memory buffer) of a user-specified number of words (up to the maximum memory buffer size) from all 15 busses after locating the trigger word in the bus serial data stream;

3. down-loads the data collected from DAS local memory to the host computer memory (a VAX-11/750) through a direct memory access (DMA) operation at 400KW/s.

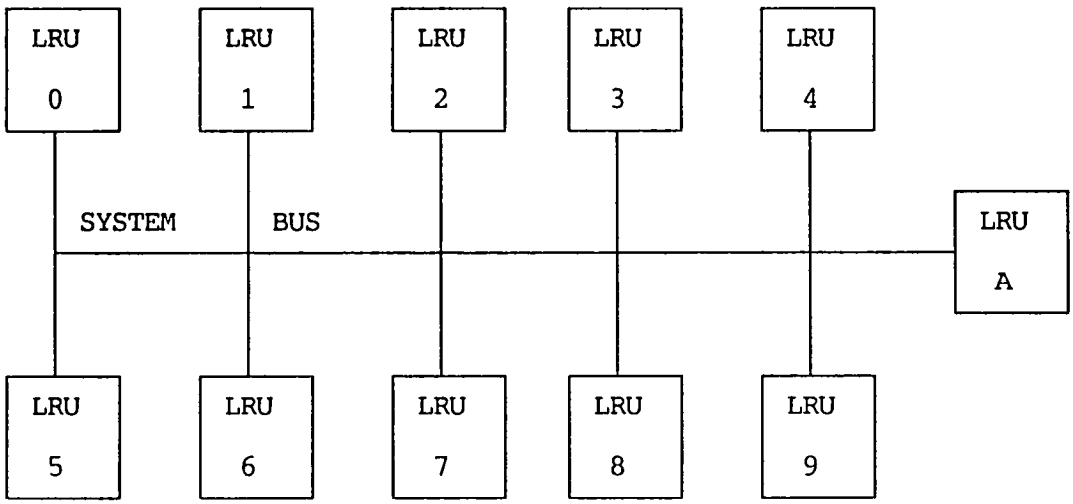
A device driver was developed for the host system through which application programs can control and integrate the data acquisition system with existing experiment control and data acquisition software.

The data acquisition system design was driven by the need to know the state of the fault tolerant test-bed at any moment during a fault injection experiment. In the FTMP test-bed, state data are transmitted on the redundant system bus every 40ms. Thus, by monitoring the system bus and triggering the DAS on the appropriate "trigger" word, it is possible to acquire these data.

The composition of this document is as follows: the following section describes the FTMP environment, i.e., the FTMP test-bed, the CTA, and 1553 data acquisition paths. The next section describes the DAS hardware and its interface to the FTMP-VAX environment. Another section describes the device driver functions and the interface to application programs (a simple example program is included in appendix B). The last section compares the new data acquisition path with the previous ones and offers some remarks about the possible use of the new capabilities. In appendix A the system bus data rates for read and write operations are estimated.

## FTMP SYSTEM ENVIRONMENT DESCRIPTION AND BACKGROUND

The Fault-Tolerant Multi-Processor system is a test-bed used for fault-tolerant systems validation experiments. Validation experiments include fault-free performance and fault injection experiments which comprise the two major subdivisions of the validation methodology presented in reference 1. More on fault-free experiments can be found in reference 2. Fault injection experiments on FTMP are described in references 3 and 4. The FTMP system is described in reference 5.




LRU = Line Replaceable Unit

Figure 1a. FTMP physical configuration

Figure 1a shows the physical configuration of FTMP. There are 11 LRUs which are interconnected by a redundant system bus. Each LRU contains (see fig. 1b) a CPU, memory management unit (MMU), 8K cache RAM, interval timer, 8K PROM, system bus interface (SBI), 16 control and communication registers (CCR), 1553 I/O port, 16K system RAM, and a real-time clock (RTC).

A processor (i.e., computer) is implemented on each LRU with the following components: CPU, MMU, cache RAM, timer, and PROM.

At system restart, different components of the LRUs (e.g., processors) are organized in groups of three (called triads, see fig 1c) to provide the redundancy required to tolerate a single fault anywhere in the system. The triads are tightly synchronized, i.e., all the processor components of a

processor triad execute the same instruction of the same program at the same clock cycle. (There are special hardware and software components in the system to support these functions, see refs 5 & 7.) Thus synchronized, the processor triads output three copies of the results of any computation/operation on the redundant system bus triad. The copies are then voted bit by bit to mask any errors that might occur due to a hardware fault on any of the triad components.

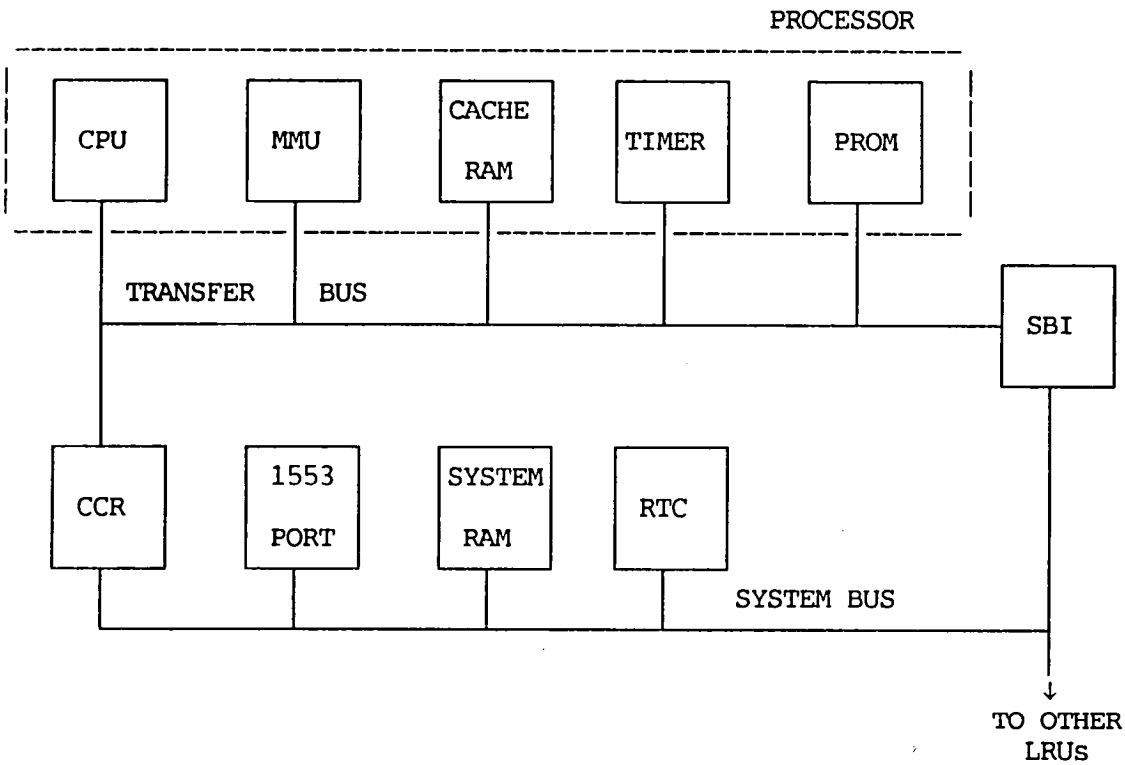



Figure 1b. LRU components

The transfer bus inside an LRU is a 16-bit parallel bus. The CCRs are write-only from the system bus and read-only from the transfer bus. The CCRs provide CPU control functions and inter-processor triad communications. Through these registers a processor can reset, interrupt, and provide the triad identification code to another processor triad. (The triad identification code is used for the poll sequence mechanism of the system bus and for system bus address decoding functions, see reference 5.)

The system bus is a composite of 4 different redundant serial busses. These busses are the Poll bus (P bus), the Transmit bus (T bus), the Receive bus (R bus), and the Clock bus (C bus). There are five busses of each P, T, R, and C

bus, of which three are being used at any given time (a bus triad) with one exception, the C bus. The real-time clocks and the C bus are configured as quads (four units active) to implement the fault tolerant clock.

The content of the C bus triad is a Non-Return to Zero (NRZ) 1MHz square wave.

The P bus triad is used by a CPU triad to request control of the T and R bus triads to access other components of the system. The P bus data rate is 1MHz and the data format is NRZ.

The T bus triad is used by a processor triad to transmit read/write commands to system memory triads, 1553 ports, the fault tolerant real-time clock, and the CCRs. The data rate is 8MHz and it is transmitted as a series of pulse width modulated pulses.

The R bus is used by memory triads, 1553 ports, and the real-time clock to respond to processor triad read commands transmitted through the T bus triad. The R bus triad data rate and format are the same as those of the T bus.

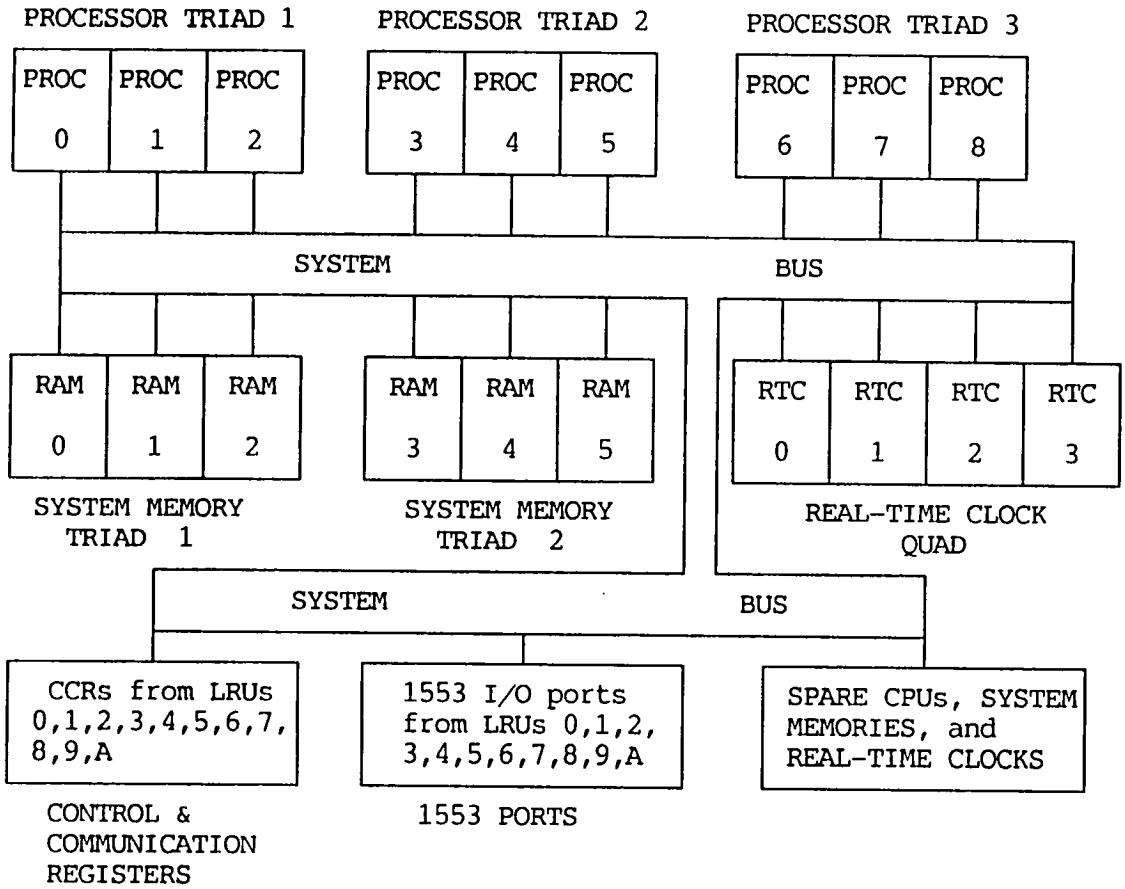



Figure 1c. FTMP configuration

One processor (processor A) in FTMP is used as a "master" processor during system restart. This "master" processor is used to load system memory with the application software and to issue initial configuration commands, e.g., which processors are going to be in triad 1, triad 2, and triad 3.

After system restart, the master processor is used for data acquisition and software debugging purposes so that it is never part of a triad.

The FTMP is interfaced to a VAX minicomputer through several hardware subsystems, which comprise the data acquisition paths. A high level block diagram showing the FTMP-VAX hardware environment is shown in figure 2.

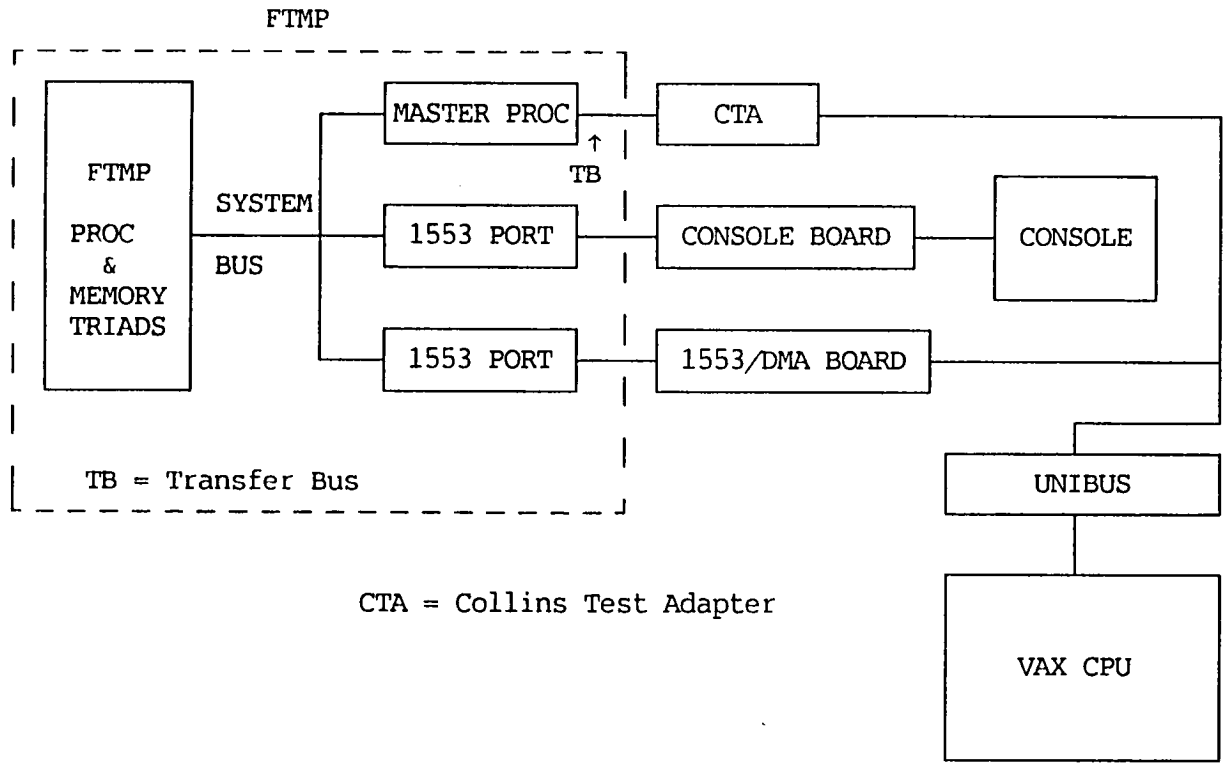



Figure 2. FTMP-VAX environment.

Each major component of the FTMP-VAX environment is represented by a block in fig. 2. The VAX CPU accesses peripherals through the UNIBUS, a 16-bit data bus. The console is a Hewlett Packard (HP) 2645A terminal and CTA is a hardware interface (bus protocol converter/translator) between the UNIBUS and master processor transfer bus.

The CTA hardware allows software running in the VAX host to read and/or write to any location in the master-processor cache memory. Unfortunately, all the data is stored in global memory. Thus, to access global memory, a program is loaded and executed in the master processor (through the CTA interface). The master-processor program can access global memory through the system bus and can communicate with the software running in the host computer through the CTA interface. All I/O operations done through the CTA interface must be programmed I/O operations (i.e. one word per read/write operation) and delay loops must be incorporated in the host software to allow it to wait until the master-processor program completes each operation.

The console board translates console commands from the HP terminal format to 1553 format which are then read by the FTMP (the bus master, see ref. 6) and

acted upon. The console board also translates FTMP status information from 1553 format to HP format so that it can be displayed on the console's screen.

The 1553/DMA board translates between 1553 format to UNIBUS format and performs read/write DMA operations to the VAX main memory. During fault injection experiments, the FTMP-VAX handshaking and fault injection data are exchanged through this interface.

As mentioned above there are two data acquisition paths from FTMP to the VAX, through the CTA and the 1553 port-UNIBUS. The data rates for these data acquisition paths can be estimated as follows.

To transfer data through the CTA interface it is necessary to run a program in the master processor. The master processor is one of the links of this data path chain; the other links are 16-bit parallel busses (UNIBUS & transfer bus), the system bus, and the CTA protocol converter hardware (fig 2).

The throughput of a FTMP processor was measured in ref 2, and is given as 150,000 instructions/s. The number of instructions required to set up the system bus interface (a DMA controller) and to transfer data from FTMP to the VAX is approximately 20 processor assembly instructions (obtained from the program listing). Only one word can be transferred at a time using the CTA hardware. Thus, the resulting data transfer rate is 7.5KW/s (1KW/s = 1 thousand 16-bit words per second). This data rate figure for the master processor is an optimistic estimate, no account has been taken of all the delays incurred in transmission, bus arbitration, etc.

In practice, the data rate through the CTA had to be reduced to ~ 40W/s ; any faster and the UNIBUS timed out, indicating that it did not receive a response from the CTA/master processor link. The reason for the time-out is saturation of the transfer bus traffic. The CPU, the system bus interface, and the CTA protocol translator (figs 1b & 2) all require the use of the transfer bus and, usually, the system bus interface wins the arbitration battle, thus forcing the others to wait.

The other data path for acquisition is somewhat more complex than the CTA

data path. The data path component chain is:

global memory ↔ system bus ↔ processor ↔ system bus ↔ 1553 port ↔

1553/DMA board ↔ UNIBUS.

Where the "↔" symbol means bidirectional data flow. A data rate estimate of each path link follows.

A 1553 port transaction, which involves 32 data words, a command word, and a status word, requires about 700$\mu$s to complete (ref 5), for an effective data rate of 32W/700$\mu$s = 45.7KW/s between the 1553 port and the 1553/DMA board. The maximum rate between the DMA board and the UNIBUS is given as 700KW/s.

The 1553 port must be accessed through the system bus by a processor. The system bus read and write data rates were estimated at ~32KW/s and ~34KW/s, respectively. (See appendix A for the derivation of these estimates of the system bus data rates.) Full utilization of the 1553 port to output data requires 134.6% of the system bus write data rate, while full utilization of the 1553 port to input data requires 143.7% of the system bus read data rate. Therefore, the 1553 port data transfer capacity cannot be fully utilized. There is, however, a more serious limitation to this data path speed.

A program must be developed for the master processor to read data from system memory to local cache and then from cache to the 1553 port (approximately 40 instructions would be required for this). Thus, the master processor data rate is reduced to $\leq$ (150/40)KW/s = 3.75KW/s. Transfer bus traffic saturation effects would reduce the data rate even further (take the CTA case for an example).

Another consideration is that only 32 words can be transmitted through the 1553 port in one transaction; thus, longer transactions must be partitioned into several 32 word transactions. This would require more instructions to be executed in the master processor, thus slowing the data rate, and an increase in the system bus traffic (a polling sequence must be executed on every transaction) which can lead to bus traffic saturation and a further reduction in the data rate. Therefore, the maximum data rate supported by this data path seems to be approximately 2 to 3KW/s.

During fault injection experiments two 1553 transactions (a read and a write operation) are performed every 40ms to transfer handshake and fault injection data between the VAX and FTMP. Therefore, the present data rate through this data path is approximately 1600W/s, well within the path data rate capacity, but leaving little room for expansion.

A current fault injection experiment involves the study of the fault effects on the system behavior. The data requirement for this experiment is approximately 150 words. These data must be sampled at least twice every 40ms for a data rate of 7.5KW/s . It is clear that the capabilities of the CTA and 1553 data acquisition paths in the FTMP environment are inadequate to fulfill this data rate requirement.

## DATA ACQUISITION SYSTEM DESCRIPTION

A new data acquisition system path was developed and added to the FTMP environment. Figure 3 shows the FTMP environment with the data acquisition system added.

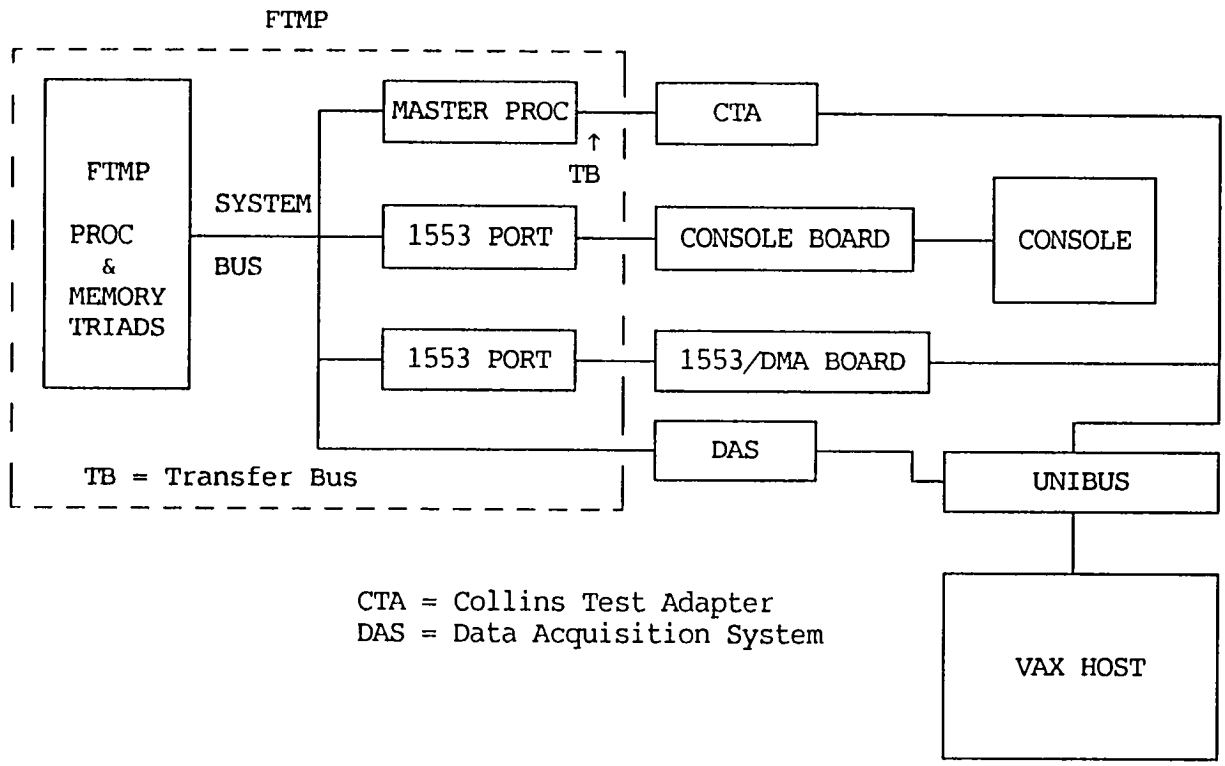



Figure 3. FTMP-VAX environment.

This path was designed with 5 characteristics in mind. These characteristics are:

1. acquire data from the target system without affecting the target system behavior.

2. transfer the acquired data as fast as possible to the host system.

3. start data acquisition as soon as previous data is transferred to the host (if repetitive acquisition is required).

4. independent operation of the data acquisition system from the host (host can be dedicated to data analysis and storage).

5. data acquisition functions should be determined by control flags/words set by the host in real-time.

The first characteristic refers to an obvious desirability in measurement systems. If the measurement system affects the target system, the measurements taken might be of limited use or worse yet, invalid and useless.

The second and third characteristics above refer to the desirability of speed in the data acquisition process.

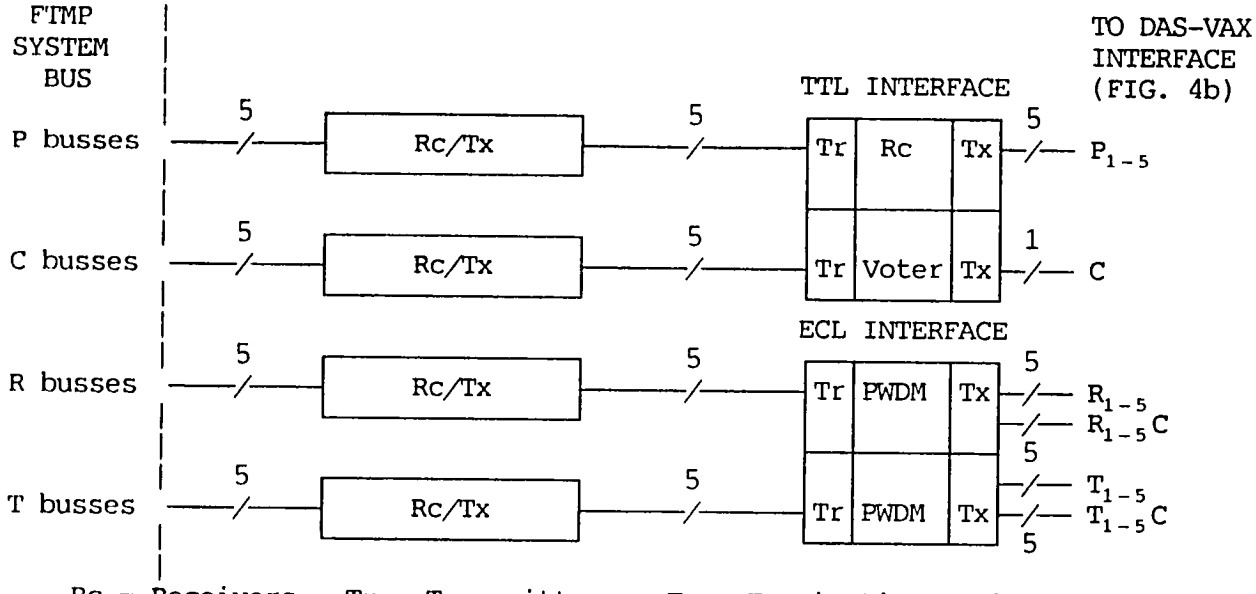
The fourth characteristic, independence from host, enhances the portability of the acquisition hardware. Also, it supports faster speeds through the implementation of acquisition and data transfer functions in hardware. Unloading the host from an I/O intensive task permits design optimizations which could not be possible if the entire operation were tightly controlled from a timeshared computer (e.g. compare a software controlled data transfer process as the CTA data path, with a data rate of 7.5KW/s, with DAS's direct memory access rate of 400KW/s).

The fifth characteristic offers some user flexibility and control in the data acquisition process. This characteristic allows the host to adjust or change the data acquisition system functions (e.g. how many words to acquire) in real-time without having to restart the experiment or modify software.

Everything of interest in FTMP, e.g., system status and configuration data, appears on the bus periodically to be processed by the operating system or application software. The DAS system monitors the system bus and can trigger on any selected word that appears on a user specified bus, be it P, R, or T bus. After triggering, the DAS stores the content of these busses for a user specified number of clock cycles in a local buffer. After storing the data, it will signal the host that it is ready to transmit data, which the host acknowledges by setting a flag, if the host is ready to accept the data. After the host flag is asserted, the DAS will DMA the data at a rate of 400KW/s to a user specified buffer in the host main memory. The present DAS local buffer size is 8KW. The buffer can be expanded to a maximum size of 64KW.

After down-loading the data to the host, the host can command the DAS to start another data acquisition cycle by setting a flag, or it can change some of the DAS parameters (e.g., the trigger word) and continue the data acquisition process.

Figures 4a and 4b present the FTMP-DAS and DAS-VAX interfaces.




Rc = Receivers , Tx = Transmitters , Tr = Termination resistors

PWDM = Pulse Width De-Modulation Circuitry

Figure 4a. FTMP-DAS interface

The T and R busses are pulse-width-modulated; thus, they must be demodulated to provide data ($R_{1-5}$ and $T_{1-5}$) and clock ($R_{1-5}C$ and $T_{1-5}C$) signals. The clock signals are then used to sample the data signals.

The P busses data signals are in a NRZ format which can be interfaced directly to DAS. The 5 C busses are voted to generate one sampling clock (labeled C in the figures) for the $P_{1-5}$ signals.

After demodulation and buffering, the signals are transmitted to the DAS. Figure 4b, presents a block diagram of DAS and its interface to the VAX host.

The data and clock signals go to a multiplexer (mux). The mux picks the signal to be examined for the trigger word and the corresponding clock to sample this signal. Data signals also go to a register/latch where they are

stored for one clock cycle, thus presenting a stable input for the duration of the memory write cycle. The signals are not latched and written into memory until the trigger word has been detected.

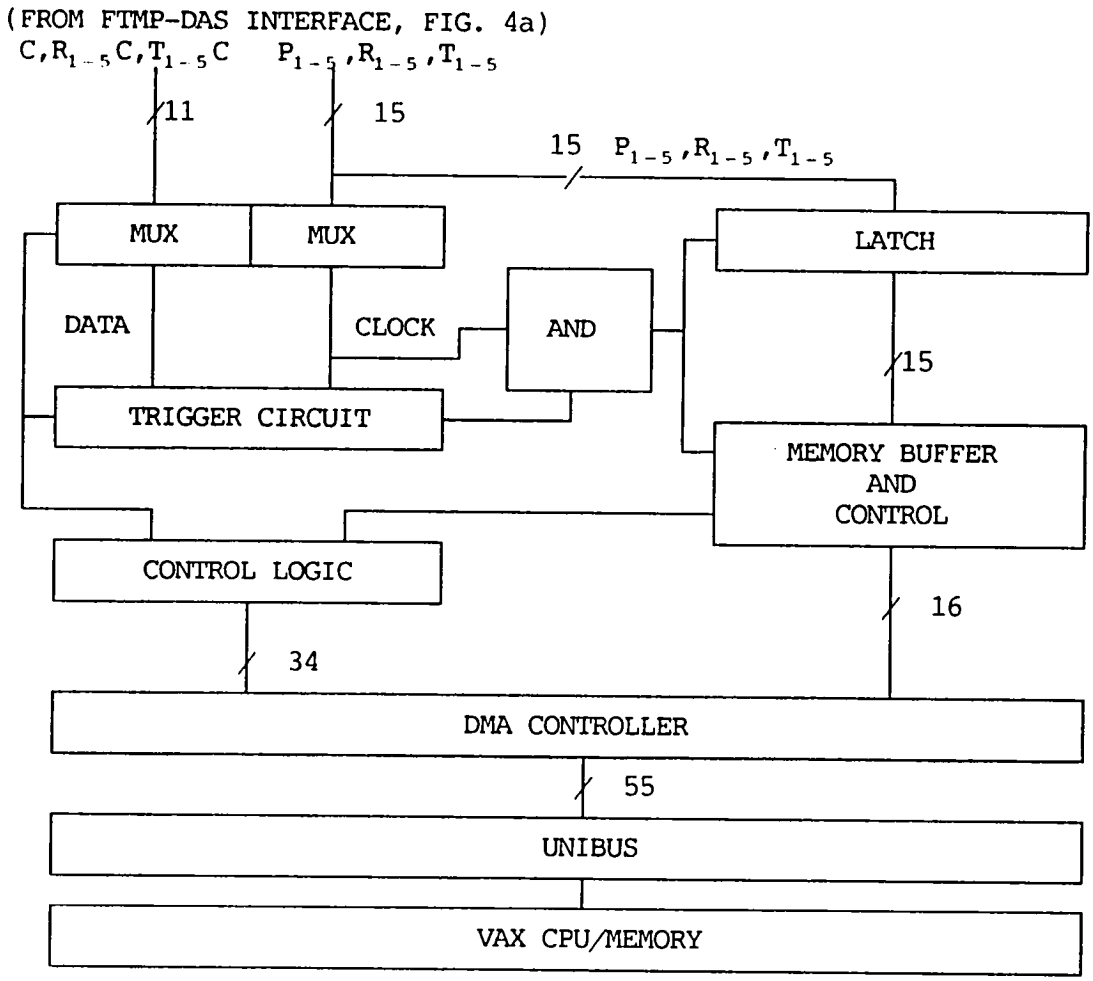



Figure 4b. DAS-VAX interface block diagram.

After trigger detection, the data is written sequentialy into the DAS memory buffer while its address in local memory (starting from 0) is compared to the number of data words specified by the user. When both are equal, the DAS will signal the host that is ready to down-load data. At this signal the host can respond in one of three ways:

1. down-load the data and reset DAS.

2. down-load the data and start acquisition again (with the same DAS parameters).

3. down-load the data, change DAS parameters, and start acquisition.

The data is down-loaded at a rate of 400KW/s. Changes in DAS parameters take approx 8$\mu$s, while starting the acquisition process takes < 200ns (these times represent the delay after the host has issued the appropriate instructions).

One word of data is stored every clock cycle of the sampling clock which is selected by the user. The data is stored in serial format, i.e., one DAS word represents one time slice (a sample) of the bus (fig 5a). Each bit in the DAS word contains the data present on a different bus of FTMP at the sampling instant (fig 5b).

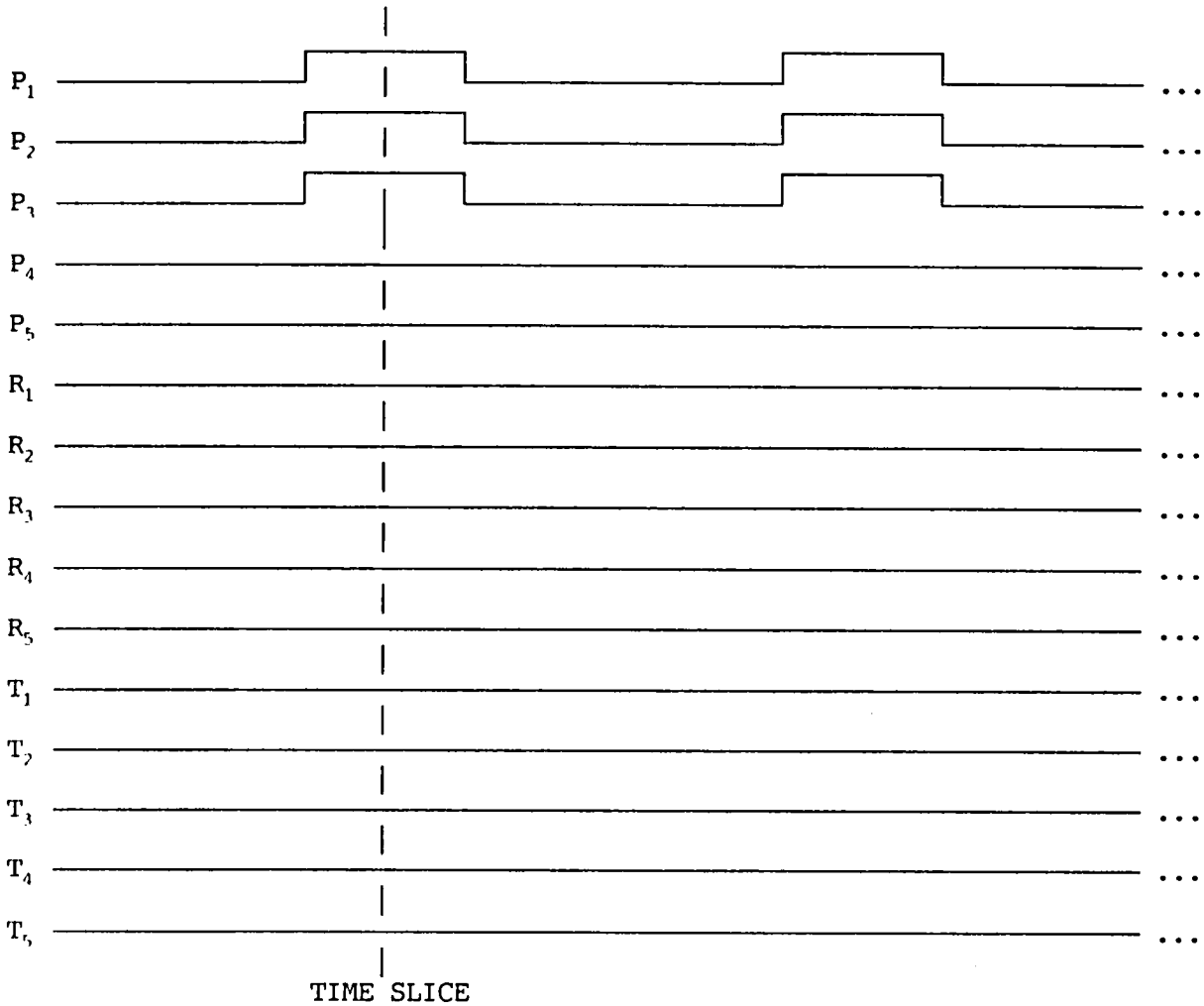



Figure 5a. A time slice of the system bus. The system is in the middle of the polling sequence.

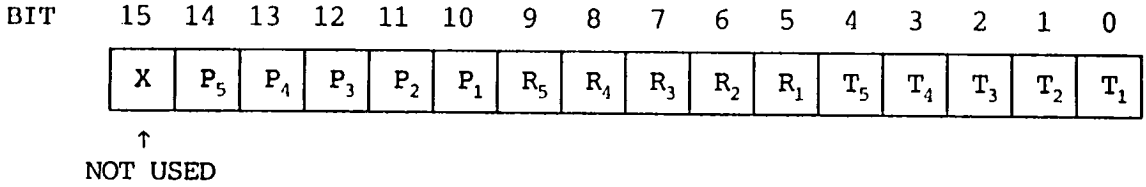



Figure 5b. Data acquisition bit assignment.

All the components of FTMP's system bus transmit data serially. Therefore, in order to acquire one 16-bit word being transmitted in a bus, 16 clock cycles must be stored in the DAS memory. Thus, 16 DAS words must be used to capture one FTMP word, but 14 additional FTMP words can be captured within those 16 DAS words, i.e., each DAS word can contain one bit of 15 different FTMP words.

As mentioned above there are a certain number of DAS parameters which the user can select (within certain bounds). These parameters are:

1. the trigger line and sampling clock selection.

2. the trigger word (16 bits)

3. the DAS word count (at present $\leq$ 8K; expandable to 64K)

The trigger line and sampling clock cannot be selected separately, i.e., if the selection for the trigger line is $R_1$, the sampling clock selection is automatically $R_1C$. This is enforced by using only one code to specify the pair ($R_1$, $R_1C$). The trigger lines and sampling clock pairs are shown in fig 6. The code which is used to select each pair is the bit number in which they reside in the DAS word.

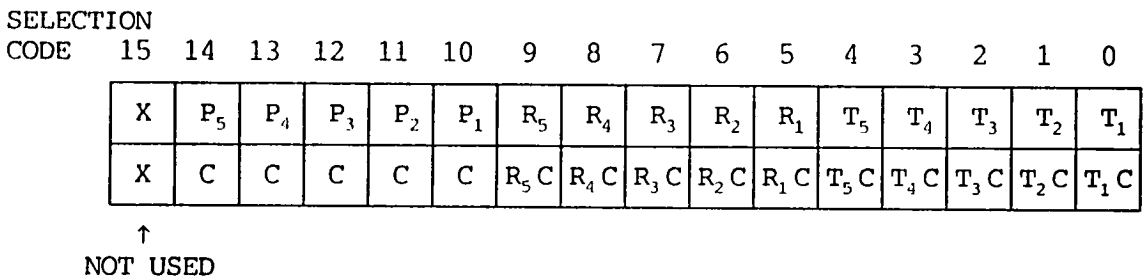



Figure 6. Trigger lines and sampling clock pairs.

Observe that only one clock is used for the P busses while the T and R busses each have their own corresponding sampling clocks. The reason for this is that the 5 C busses are phase locked, i.e., they are all synchronized. Thus it does not matter which C bus is used to sample the P bus transmissions (which are also synchronized). In order to assure a sampling clock for the DAS, the 5 C busses are voted in the FTMP-DAS interface, and the voted clock is used by the DAS to sample $P_1$ through $P_5$.

In contrast, the time skew between the T (or R) busses transmissions are not guaranteed to be less than half a bit period (62.5ns), i.e., they are not synchronized. Thus, the appropriate sampling clock must be used to sample a signal for the trigger word.

In order to observe timing relationships between data signals, all the signals are sampled by the selected clock. Thus, any time skews between the elements of the T (or R) bus triads can be detected. This property is very useful for timing analysis and studies of fault injection effects on system behavior.

As mentioned above, the DAS works independently from the host VAX though it receives control information through several control flags and registers. A program that provides a software interface to the hardware and hides all the hardware complexity from the user was developed. The next section provides a description of the DAS software interface.

## DAS SOFTWARE INTERFACE

A control and data flow graph for the VAX-DAS interface is shown in figure 7. This flow graph represents the information and control flow from the application programs to DAS. The device driver program translates application program commands into the bit patterns understood by the DAS hardware. When requested by the applications program (with a command), the device driver will automatically set up the DMA transfer that will write the acquired data to a application program data buffer or it will load DAS control registers with information stored in another application program buffer.

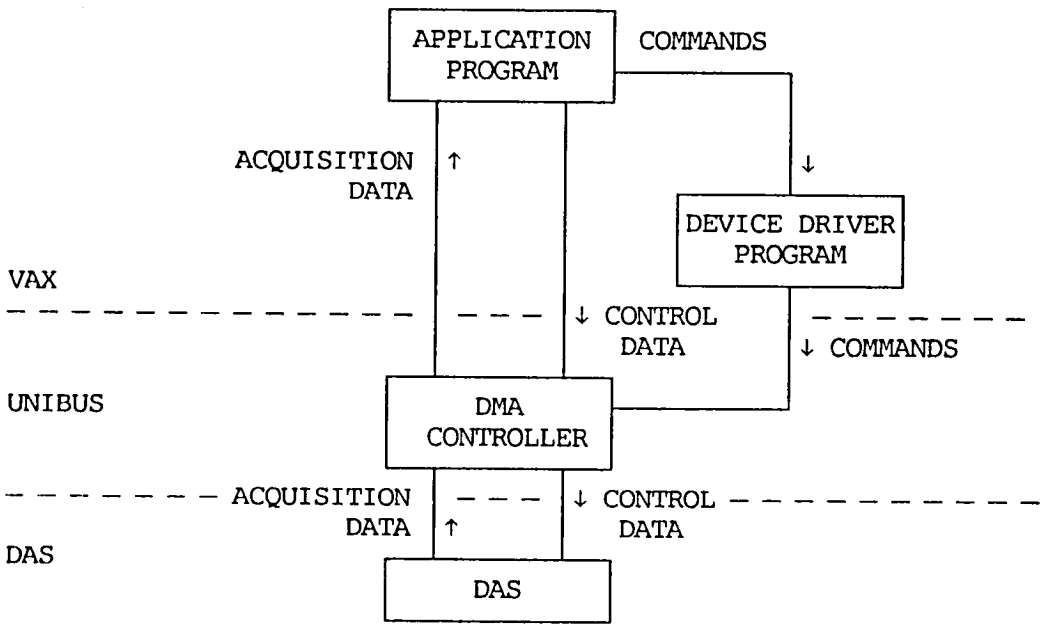



Figure 7. Control and data flow graph of the VAX-DAS interface.

There are several commands that an application program can issue. Each command is related to one of the functions that DAS can perform: down-load data to the VAX, start acquisition, reset, get status information, and load DAS control registers.

When a "load DAS parameters" command is issued by the application program, the device driver sets up a DMA controller. The controller reads the application program data buffer where the DAS parameters are stored, and loads them into DAS control registers.

At reception of the command "start DAS", the driver sets a flag that indicates to the DAS hardware to start acquisition immediately.

The application program can access a flag, set by DAS to indicate its readiness to down-load data, through the command "get csr" (csr = control and status register). The flag is contained in bit 10 (word bits numbered from 0 to 15) of the csr register. Other bits are used for debugging purposes.

If the command is "read DAS data", the driver will set up the DMA controller to down-load the data from the DAS local buffer to an application program's buffer in the host main memory subsystem (if DAS is ready).

The command "reset DAS" immediately stops DAS from whatever it is doing and resets it. This command does not reset DAS control registers, so that a start command can be issued without having to reload the control registers.

The VAX Queued Input/Output (QIO) system service is utilized to send a command to the device driver program. (For information about QIO system service refer to Digital Equipment Corporation manuals.) The code in the driver has been streamlined and optimized to give the fastest performance possible. For most experiments, the QIO interface performance should be sufficient. (The slower commands are the "load DAS parameters" and "read DAS data", which take approx 20~50$\mu$s to initiate the indicated operation.)

The format of the QIO service and the control parameters for the different DAS commands are shown in tables 1, 2A, and 2B.

TABLE 1. QIO FORMATS FOR DAS COMMANDS[a,b]

| COMMANDS | QIO FORMAT |
|---|---|
| | |
| RESET DAS | $QIOW_S ,CHAN,#IO$_RESET,IOSB |
| LOAD DAS PARAMETERS | $QIOW_S ,CHAN,#IO$_LOAD,IOSB,-<br>,,P1=TRIGGER,P2=#6 |
| START DAS | $QIOW_S ,CHAN,#IO$_START,IOSB |
| GET CSR[c] | $QIOW_S ,CHAN,#IO$_GET_CSR,-<br>IOSB |
| READ DAS DATA | $QIOW_S ,CHAN,-<br>#IO$_READ_BUFFER,IOSB,-<br>,,P1=BUFFER,P2=#BYTES |

[a] The dash ("-") indicates that the statement continues on the next line.

[b] See Appendix B: An Example Program.

[c] The csr is returned in the second word of the iosb variable, see table 2A.

TABLE 2A. DAS QIO COMMAND PARAMETERS

| PARAMETER | DESCRIPTION |
|---|---|
| | |
| CHAN | 16-BIT WORD THAT CONTAINS A CHANNEL NUMBER USED BY THE QIO SERVICE TO DETERMINE WHICH DEVICE IT SHALL ADDRESS (SEE EXAMPLE IN APPENDIX B) |
| IOSB | 2 32-BIT WORDS WHICH ARE USED BY THE QIO SERVICE TO RETURN STATUS INFORMATION (A 0 IN THE LEAST SIGNIFICANT BIT OF THE FIRST WORD SIGNALS AN ERROR) |
| IO$_ | INDICATES A FUNCTION CODE, I.E., A HEX NUMBER WHICH INDICATES WHICH COMMAND SHOULD BE EXECUTED |
| IO$_LOAD = 000B | LOAD DAS PARAMETERS (LOAD CONTROL REGISTERS) |
| IO$_START = 001A | START DAS |
| IO$_RESET = 0024 | RESET DAS |
| IO$_GET_CSR = 001B | GET CSR (GET STATUS INFO) |
| IO$_READ_BUFFER = 000C | READ DAS DATA (DOWN-LOAD ACQUISITION DATA) |

TABLE 2B. QIO COMMANDS: P1 AND P2 PARAMETERS

| COMMAND[a] | P1 AND P2 PARAMETER DESCRIPTION |
|---|---|
| | |
| LOAD DAS PARAMETERS | P1 = ADDRESS OF THE APPLICATION PROGRAM BUFFER WHERE THE PARAMETERS ARE STORED. THE FORMAT OF THIS BUFFER IS AS FOLLOWS:<br>(3 CONSECUTIVE 16-BIT WORDS)<br>FIRST WORD: TRIGGER LINE SELECTION CODE[b]<br>SECOND WORD: TRIGGER WORD[b]<br>THIRD WORD: NUMBER OF DAS WORDS TO ACQUIRE[b]<br><br>P2 = NUMBER OF BYTES IN BUFFER (P2 = 6) |
| READ DAS DATA | P1 = ADDRESS OF THE APPLICATION PROGRAM BUFFER TO RECEIVE THE DATA (THE BUFFER SIZE MUST BE EQUAL OR LARGER THAN THE "NUMBER OF DAS WORDS" PARAMETER SPECIFIED IN THE THIRD WORD OF THE "LOAD DAS PARAMETERS" COMMAND BUFFER)<br><br>P2 = BUFFER SIZE IN BYTES (BYTES = # OF 16-BIT WORDS X 2) |

[a] Start DAS, reset DAS, and get csr commands do not require P1 or P2 parameters

[b] See previous section for a description of its function

A short VAX assembly language program that executes a data acquisition cycle is listed in appendix B. The fully commented listing contains all the information required to program the DAS.

The program shown in the appendix can be used as shown, i.e. without modifications, to read and list (on the computer terminal screen) data acquired from the FTMP system bus.

## CONCLUDING REMARKS

To provide a point of comparison between DAS and the other data acquisition paths available in the FTMP environment, an estimate of the equivalent data rate of DAS is computed below.

The present size of the DAS local buffer is 8192 16-bit words (expandable to 65535 16-bit words). At 8MHz (the data rate of the T and R busses) this buffer will be filled in 1.024ms. Add 50$\mu$s delay to send the "read DAS data" command. Add the time required to unload the 8192 data words at a rate of 400KW/s (8192W/400,000W/s = 20.48ms). Add the time required to issue a start command (approx 10$\mu$s). The total time delay from the start to unloading the data adds up to approx 21.564ms . Dividing the number of words transferred by the time required to transfer them (8192W/21.564ms) produces an estimated data rate of 379KW/s. This data rate is at least 8 times larger than that of any other data acquisition path available in the FTMP environment.

The added capabilities to the FTMP data acquisition environment should provide the necessary functions to support a new range of experiments which are geared toward determining the effects of faults on FTMP's system operations.

## APPENDIX A

## Estimation of the System Bus Data Rates.

The system bus data rates for read and write operations are estimated using published data on the FTMP's performance.

The information in refs 5 and 8 is used to estimate the system bus data rate for read operations. To obtain control of the system bus, a processor triad must win the polling procedure. From ref 8 it can be computed (table 3, p 29) that the mean wait time to start the polling procedure is 9.87$\mu$s (bus is busy 47% of the time a request is made with an average wait for a free bus of 21$\mu$s). To start the polling procedure it must transmit 12 bits in the P bus (@ 1MHz) for an average delay of 14.58$\mu$s (taking into account the delays involved when it losses the polling sequence; 8% of the time). Then it must transmit the read command through the T bus, 24 bits @ 8MHz, for a total time of 3$\mu$s . The average response delay after the command transmission is 2$\mu$s. Transmission of the data itself (1 word) takes 2$\mu$s.

Subsequent read commands are transmitted in parallel with the previous command response. Thus, the next read response (though still 2$\mu$s after the read command word is transmitted) comes only 8 bit times (1$\mu$s) after the previous response (see fig 3.4 in ref 5). Read responses cannot be packed closer than this.

Thus, the total data transmission delay for one word transaction is (9.87 + 14.58 + 3 + 2 + 2)$\mu$s or 31.45$\mu$s . The effective system bus data rate for a read operation is then 31.8KW/s .

For a transaction of two or more words, the 1$\mu$s time delay between read responses must be included to estimate the data rate.

The data rate equation for this case (tightly packed blocks of n words, n > 1) is:

$$\text{data rate} = n \times [29.45 + n * 3]^{-1} \times 10^{6} \text{ W/s} , n > 1 .$$

The term 29.45 (in $\mu$s) above is the sum of the polling time (24.45$\mu$s), the transmission of the first command word (3$\mu$s), and the wait time for the first

response (2$\mu$s). The constant in the term, n * 3, accounts for the 2$\mu$s for each data word transmission and 1$\mu$s for the idle time between data word transmissions (n is the number of words transmitted).

The system bus data rate for write operations can be computed more easily. The polling sequence time is 24.45$\mu$s, the command transmission time is 3$\mu$s (24 bits), and the data transmission time is 2$\mu$s for a total of 29.45$\mu$s for a write operation. Thus, the system bus data rate for write transactions is 33.96KW/s.

## APPENDIX B

## An Example Program

```
;       Author:  Peter A. Padilla

; define the command codes for the QIO system service from values defined
; in the system macro $iodef
;
        $iodef
io$_load = io$_writepblk          ;"load DAS parameters" command
io$_start = io$_setchar           ;"start DAS" command
io$_reset = io$_rewind            ;"reset DAS" command
io$_read_buffer = io$_readpblk    ;"read DAS data" command
io$_get_csr = io$_sensechar       ;"get csr" command
io$_boom = io$_writevblk          ;terminal I/O codes
io$_buffer = io$_ttyreadpall!io$m_timed!io$m_purge
;
; define the number of words to down-load from DAS to the host
;
count_word = 50
no_of_bytes = 2 * count_word

; define data area, allocate space for the buffer and variables

        .psect   data_1,noexe,word

chan:    .word    0
chan_2:  .word    0
loop:   .word   0
        .align word

; DAS parameters buffer area

trigger: .word    0               ;Trigger line (T1)
        .word    ^xAAAA           ;Trigger word
        .word    count_word       ;number of words
```

```
; terminal I/O messages and buffer areas

devnam:  .ascid   /das0:/
terminal: .ascid /sys$output/
error:   .ascii   /read dr11-w eir instead of the csr/
length = .-error
error_2: .ascii /data acquisition timed out/
length_2 = .-error_2
error_4: .ascii /status b is not being reset/
length_4 = .-error_4
outbuf:  .blkb    800
outl = .-outbuf
outlen:  .long    outl
         .address outbuf
control: .ascid   / R0= !XL  R1= !XL /
control_2: .ascid /data = !XW counter = !XW /
msg1:    .ascii   /Enter trigger word [AAAA]:/
length1 = .-msg1
msg2:    .ascii   /Enter trigger line number [0000]:/
length2 = .-msg2
ouuf:    .blkb    800
oul = .-ouuf
table:   .byte    0,1,2,3,4,5,6,7,8,9,^xA,^xB,^xC,^xD,^xE,^xF ;hex table

; buffer to receive down-loaded data from DAS

.align word
buffer:  .blkw    65536

; variable to receive status information from the DAS

         .psect   data_2,noexe,long

iosb:    .blkl    2
```

```
; code begins

        .psect   dastest,exe,byte

        .entry   dastest,^M<r2,r3,r4,r5>
        clrl     r10
        $assign_s terminal,chan_2      ;ASSIGN TERMINAL CHANNEL
        $assign_s devnam,chan          ;ASSIGN DAS CHANNEL
        $qiow_s  ,chan,#io$_reset,iosb      ;RESET DAS
        bsbw     ask                   ;ASK THE USER
        $qiow_s  ,chan,#io$_load,iosb,,,p1=trigger,p2=#6 ;LOAD DAS
;PARAMETERS (PARAMETERS ARE STORED IN THE 6 BYTES LONG BUFFER "TRIGGER")
        blbs     iosb,2$          ;IF NO ERROR GOTO 2$
        bsbw     print_iosb       ;PRESENT STATUS INFO AT TERMINAL
        brw      20$
2$:     $qiow_s  ,chan,#io$_start,iosb      ;START DAS COMMAND
        blbs     iosb,10$               ;IF NO ERROR GOTO 10$
        bsbw     print_iosb             ;PRESENT STATUS INFO
        brw      20$
10$:    bsbw     wait_for_das           ;WAIT FOR DAS TO SIGNAL
; ITS READINESS TO DOWN-LOAD DATA
        blbc     r0,20$                 ;IF ERROR CONDITION EXISTS
; GOTO 20$
        $qiow_s  ,chan,#io$_read_buffer,iosb,,,p1=buffer,p2=#no_of_bytes
; READ DAS DATA (STARTS DMA OPERATION)
        blbc     iosb,20$               ;IF ERROR GOTO 20$
        bsbw     print_data             ;SEND DATA TO THE TERMINAL
; SCREEN
20$:    $qiow_s  ,chan,#io$_get_csr,iosb    ;GET CSR COMMAND
        bsbw     wait_for_b             ;IF STATUS B IS SET, WAIT!
; IT IS SENDING DATA
        $qiow_s  ,chan,#io$_reset,iosb      ;RESET DAS
        $exit_s
```

```
;*************************************************************************
wait_for_das:                          ;subroutine to wait for
; DAS to set the readiness flag
        clrl    r10
10$:    incl    r10                    ;timeout loop (to detect
;                                       hardware problems)
        cmpl    r10,#20000             ;timeout value
        bgtru   40$                    ;timeout occur
        $qiow_s ,chan,#io$_get_csr,iosb   ;get csr command
        blbc    iosb,20$               ;if error goto 20$
        blbs    iosb+4,30$             ;if other error goto 30$
        bbc     #10,iosb+4,10$         ;if DAS flag = low ,goto 10$
        movl    #1,r0                  ;r0=1 no error status
        rsb
20$:    bsbw    print_iosb             ;print status info
        brw     back
30$:    bsbw    print_error            ;print "wrong register read"
;                                       error message
        brw     back
40$:    bsbw    print_error_2          ;print timeout error message
back:   $qiow_s ,chan,#io$_reset,iosb     ;after error, reset DAS
        clrl    r0                     ;r0=0 error status
        rsb

wait_for_b:                            ;wait until DAS finishes
; down-loading data before doing something else (this routine is required only
; if there is nothing else to do)
        clrl    r10
10$:    incl    r10                    ;timeout loop
        cmpl    r10,#2000
        bgtru   40$
        $qiow_s ,chan,#io$_get_csr,iosb   ;get csr command
        blbc    iosb,20$               ; error conditions
        blbs    iosb+4,30$
        bbs     #10,iosb+4,10$         ;if DAS flag = 1 , goto 10$
```

```
; down-loading in process
        movl    #1,r0                   ;if DAS flag = 0, no error
        rsb
20$:    bsbw    print_iosb
        brw     back_b
30$:    bsbw    print_error             ; wrong register was read
        brw     back_b
40$:    bsbw    print_error_4           ;DAS flag is not being
;                                        cleared, hardware fault
back_b: $qiow_s ,chan,#io$_reset,iosb       ;reset DAS
        clrl    r0                      ; signal error condition
        rsb

; The next subroutines send information to the terminal screen.

print_iosb:
        $fao_s  control,outlen,outlen,p1=iosb,p2=<iosb+4>
        $qiow_s ,chan_2,#io$_boom,iosb,,,p1=outbuf,p2=#outl,p4=#^x30
        blbs    iosb,10$
        $exit_s
10$:    rsb

print_data:
        movl    #1,r9
        clrl    r10
        clrl    r11
        movaw   buffer,r10
5$:     movw    (r10),r11
        $fao_s  control_2,outlen,outlen,p1=r11,p2=r9
        $qiow_s ,chan_2,#io$_boom,iosb,,,p1=outbuf,p2=#outl,p4=#^x30
        blbs    iosb,10$
        $exit_s
10$:    incw    r9
        addl    #2,r10
        cmpl    r9,#count_word
        blequ   5$
        rsb
```

```
print_error:
        $qiow_s  ,chan_2,#io$_boom,iosb,,,p1=error,p2=#length,p4=#^x30
        blbs     iosb,10$
        $exit_s
10$:    rsb

print_error_2:
        $qiow_s  ,chan_2,#io$_boom,iosb,,,p1=error_2,p2=#length_2,p4=#^x30
        blbs     iosb,10$
        $exit_s
10$:    rsb


print_error_4:
        $qiow_s  ,chan_2,#io$_boom,iosb,,,p1=error_4,p2=#length_4,p4=#^x30
        blbs     iosb,10$
        $exit_s
10$:    rsb

; The next subroutine request information from the user.

ask:
;   qios sends prompt to the screen and waits for answer
;
        $qiow_s  #2,chan_2,#io$_buffer,iosb,,,p1=ouuf,p2=#oul,-
                 p3=#120,p4=#0,p5=#msg1,p6=#length1
;
;   input the trigger word above
;
        blbs     iosb,10$
        $exit_s  iosb
10$:    bsbw     convert
        movw     r11,<trigger+2>        ;r11 = result of conversion
        $qiow_s  #2,chan_2,#io$_buffer,iosb,,,p1=ouuf,p2=#oul,-
                 p3=#120,p4=#0,p5=#msg2,p6=#length2
;
```

```
; qio asks for the trigger line code.
;
        blbs    iosb,20$
        $exit_s iosb
20$:    bsbw    convert
        movw    r11,trigger
        rsb
;
; The next routine converts ascii numbers to hex numbers
;
convert:
        clrl    r8
        clrl    r9
        clrl    r10
        clrl    r11
        cmpb    ouuf,#^X39          ;input in ouuf
        blequ   5$
        addb2   #9,ouuf
5$:     bicb3   #^XF0,ouuf,r8
;
        cmpb    <ouuf+1>,#^X39
        blequ   10$
        addb2   #9,<ouuf+1>
10$:    bicb3   #^XF0,<ouuf+1>,r9
;
        cmpb    <ouuf+2>,#^X39
        blequ   20$
        addb2   #9,<ouuf+2>
20$:    bicb3   #^XF0,<ouuf+2>,r10
;
        cmpb    <ouuf+3>,#^X39
        blequ   30$
        addb2   #9,<ouuf+3>
30$:    bicb3   #^XF0,<ouuf+3>,r11
;
```

```
        ashl    #4,r8,r8
        ashl    #4,r10,r10
        bisl2   r8,r9
        bisl2   r10,r11
        ashl    #8,r9,r9
        bisl    r9,r11
        rsb

.end dastest
```

## REFERENCES


1. "VALIDATION METHODS RESEARCH FOR FAULT TOLERANT AVIONICS AND CONTROL SYSTEMS - WORKING GROUP MEETING II," NASA CP-2130, October 1979.

2. E. Clune, Z. Segall, and D. Siewiorek, "FAULT FREE BEHAVIOR OF RELIABLE MULTI-PROCESSOR SYSTEMS: FTMP EXPERIMENTS IN AIRLAB," NASA CR-177967, August 1985.

3. J.H. Lala, and T.B. Smith III, "DEVELOPMENT AND EVALUATION OF A FAULT TOLERANT MULTI-PROCESSOR (FTMP) COMPUTER, VOLUME III, FTMP TEST AND EVALUATION," NASA CR-166073, May 1983.

4. G.B. Finelli, "CHARACTERIZATION OF FAULT RECOVERY THROUGH FAULT INJECTION ON FTMP," IEEE Trans. on Reliability, VOL. R-36, NO. 2, June 1987, p 164-170.

5. T.B. Smith III and J.H. Lala, "DEVELOPMENT AND EVALUATION OF A FAULT TOLERANT MULTI-PROCESSOR (FTMP) COMPUTER, VOLUME I, FTMP PRINCIPLES OF OPERATION," NASA CR-166071, May 1983.

6. "MIL-HDBK-1553, MULTIPLEX APPLICATIONS HANDBOOK," Department of Defense, November 1984.

7. J.H. Lala and T.B. Smith III, "DEVELOPMENT AND EVALUATION OF A FAULT TOLERANT MULTI-PROCESSOR (FTMP) COMPUTER, VOLUME II, FTMP SOFTWARE," NASA CR-166072, May 1983.

8. K.G. Shin, M.H. Woodbury, and Y. Lee, "MODELING AND MEASUREMENT OF FAULT TOLERANT MULTI-PROCESSORS," NASA CR-3920, August 1985.

NASA

# Report Documentation Page

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| NASA TM-100636 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| FTMP Data Acquisition Environment | July 1988 |
| | 6. Performing Organization Code |

| 7. Author(s) | 8. Performing Organization Report No. |
|---|---|
| Peter A. Padilla | |
| | 10. Work Unit No. |
| | 506-46-21-05 |

| 9. Performing Organization Name and Address | 11. Contract or Grant No. |
|---|---|
| NASA Langley Research Center<br>Hampton, VA 23665-5225 | |
| | 13. Type of Report and Period Covered |
| 12. Sponsoring Agency Name and Address | Technical Memorandum |
| National Aeronautics and Space Administration<br>Washington, DC 20546-0001 | 14. Sponsoring Agency Code |

15. Supplementary Notes

16. Abstract

The Fault-Tolerant Multi-Processor (FTMP) test-bed data acquisition environment is described. The performance of two data acquisition devices available in the test environment are estimated and compared. These estimated data rates are used as measures of the devices' capabilities.

A new data acquisition device was developed and added to the FTMP environment. This path increases the data rate available by approximately a factor of 8, to 379 KW/S, while simplifying the experiment development process.

| 17. Key Words (Suggested by Author(s)) | 18. Distribution Statement |
|---|---|
| Fault-Tolerance<br>Data Acquisition<br>Fault-Tolerant Multi-Processor (FTMP) | Unclassified-Unlimited<br><br>Subject Category 33 |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 37 | A03 |