THE EFFICIENT USE OF A VAX COMPUTER IN THE REAL-TIME TELEMETRY ENVIRONMENT

Robert B. Robbins
Sangamo-Weston Incorporated
Data Systems Division
Post Office Box 3041
Sarasota, Florida 33578

ABSTRACT

The use of a Digital Equipment Corporation VAX computer under the VMS operating system, in a real-time telemetry environment, brings with it many advantages. These advantages pertain to its ability to handle real-time telemetry processing in an efficient and relatively straight forward manner.

The author will use the TELSET, TELDAX and TELFOR telemetry software systems as the basis for demonstrating the techniques which have allowed the real-time telemetry user to take advantage of a 32-bit, virtual addressing, architecture.

INTRODUCTION

Many real-time telemetry computer users believe in the need for 32-bit processing of telemetry data once it has entered the central processor. We at Sangamo-Weston agree with this belief and have successfully designed a real-time telemetry data acquisition system which efficiently uses the VAX computer under the VMS operating system.

SYSTEM SOFTWARE

The software written for this system achieves three goals. TELSET, which sets up the telemetry front-end hardware units, is written in VAX-11 MACRO. TELDAX the real-time data acquisition executive and TELFOR the telemetry data acquisition formatted storage process are written in interactive VAX-11 FORTRAN and are menu driven.

The design of the real-time portions of TELDAX addresses certain efficient uses of the VAX for parallel processing. These areas include the use of global common areas and global event flag clusters; the VMS group of system service routines for creation, activation and deletion of detached processes; and effective real-time communications

using mailboxes, flags and the $QIO system service. A basic knowledge of VMS's approach to interrupt priority level processing, use of system privileges, and kernel mode processing is necessary for efficient use of the VAX operating system in a real-time environment. The aim of this paper is to give the reader a basis for further research in the area of real-time telemetry parallel processing.

SYSTEM HARDWARE

The hardware configurations for the TELDAX series of software are flexible and expandible. The 700 series of Sangamo-Weston programmable telemetry front-end units include PCM frame/subframe synchronizers, PAM synchronizers, FM multiplex encoders, programmable word selectors, bit synchronizers, time code generators and translators, tape search units and the very powerful multiplex preprocessors with user programmable microcoded algorithms. In addition, programmable interfaces connect the telemetry front-end units to the versatile DEC Unibus. These interfaces include the buffered data channel and priority interrupt module.

The telemetry front-end can provide up to sixteen input data streams per Unibus with an aggregate input rate of 4,500,000 bits per second to the CPU. Each data stream flows through the Sangamo-Weston Model 760 Buffered Data Channel. These buffered data channels allow the user to pass blocks of data, up to 32K words long, with quadruple buffering, in a cyclic fashion.

A major advance in the preprocessing of telemetry data is the Sangamo-Weston Model 715 Multiplex Processor. This unique hardware preprocessor is microprogrammable and can perform most of the repetitive real-time processes which are required in high-speed data analysis. The 715 merges data from up to six asynchronous sources at burst rates of up to 2,000,000 words per second and can distribute the processed data to the computer and/or other devices via three separate output parts. It processes both floating point and integers and can convert telemetry data to engineering units at a ratio of 100,000 to 300,000 words per second. In addition to EU conversion and data compression a wide range of additional data processing functions are available. The 715 is directly interfaced to the above-mentioned Model 760 Buffered Data Channel and is under the direct software control of the TELDAX system when a part of the system configuration.

Our experience with data acquisition systems on the VAX indicates that the controlling factor for top throughput rates is not the rate in which data is input into the system over the Unibus, or software time spent in processing and formatting of data for output to a storage device, but the rate at which the storage device can accept the data. Too often, disk rotation and disk head movement limitations are overlooked when throughput is being

estimated prior to the start of a project. This must be considered as a major timing factor in the real-time telemetry world.

INTERRUPT PRIORITY LEVELS

The Interrupt Priority Level (IPL) method by which VMS allows processes to be run is a two-fold method. There is a 32 level set of hardware interrupt priorities and a 32 level set of software priorities. The entire 32 level set of software priorities reside in hardware interrupt priority level 0. This means that any hardware interrupt takes precedence over all software interrupts. The 32 software priority levels are separated into two groups also; regular processing - levels 0 through 15; and real-time processing - levels 16 through 31. The null process is found at software level 0 and the swapper is found at software level 16.

All real-time processing is done above software priority 16. If this rule is not followed, the user takes the chance of being out-swapped by a non-real-time process. In addition, real-time processes should remain as close to the swapper as possible so that swapping is accomplished immediately when necessary. It is important to remember that the ultimate goal of the designer of real-time telemetry processes is to remain in “current-state” at all times. This means taking all measures necessary to remain in the grasp of the CPU. There are ten ways to be removed from the current state (CPU execution) by the VAX/VMS scheduler and swapper. By removing these ten possibilities in the coding of real-time processes a process will not be out-swapped.

KERNEL MODE PROCESSING

The use of VMS system service routines allows the real-time user to efficiently process data by running in kernel mode instead of user mode during time-critical periods.

Running a real-time telemetry process demands time-efficiency for success. This means kernel mode processing. Anything less is inefficient. For this reason, when a process is running in real-time, Record Management System (RMS) calls should not be used, as RMS runs in executive mode. There are VMS system service routine replacements for RMS calls which run in kernel mode. For instance, $QIO can be used to replace TYPE, WRITE, READ, PRINT, ACCEPT and others. $ALLOC, $ASSIGN, $DALLOC and $DASSGN can be used to replace OPEN and CLOSE.

THE PROCESS LIFE CYCLE

There are three phases in the process life cycle:

- process creation
- process activation
- process deletion

Real-time processing is most efficient when process creation and process activation are treated as unique entities. This can be accomplished by creating all processes required for real-time operations prior to real-time start-up and cueing process activation by use of interrupt driven event flags. In this way the time taken to handle the creation of a process, up to five seconds per process, depending on the type of process creation being done, is not done at the time that real-time telemetry data acquisition should be going on.

CREATING A REAL-TIME PROCESS

Process creation on the VAX is handled most efficiently by use of the $CREPRC system service routine. $CREPRC will dynamically create either a sub-process or a detached process, at run time, depending on the routine setup. The choice of creating a sub-process or a detached process is heavily dependent on the requirements of the application. Sub-processes share the quota set of the main or creating process and are deleted when the creating process exits, guaranteeing that the sub-process will not remain in limbo, in the system, after the realtime program run ceases to exist. A detached process is an entity into itself, with its own set of quotas, own priority level and own UIC. A detached process, once it is created, runs under its own process control, and relies on its own program logic or externally fed information (e.g. common event flags, interrupts, mailboxes) to exit. TELDAX creates detached processes, allowing the user to leave the main-line program and do other tasks, with the ability to reenter the main-line program at a later time, if necessary. By using detached processes, the telemetry data acquisition streams (processes) are not effected by the exit from and reentrance into TELDAX.

THE CREATED PROCESS

Once the process is created, there are several activities which are a part of initializing the real-time process, prior to process activation. The basic activities of input and output file channel assignments and initialization of variables are ever present in application programs and this does not change for created processes. There are several important steps that must be accomplished to guarantee the presence of current state for each real-time process created.

In TELDAX, we communicate some dynamic information, prior to real-time processing, from the main-line process to all detached processes, via a mailbox. Therefore, we use the $CREMBX create mailbox system service routine.

The $CREMBX system service assigns a mailbox channel if a channel has not been previously assigned. The mailbox name is placed in quotes and differentiates this mailbox from others used in the system. TELDAX uses a separate mailbox to communicate with a detached process containing the system error logger. All mailboxes communicate over the same mailbox channel. Therefore, a separate channel variable need not be declared for each mailbox used.

Common event flags serve as a quick and efficient way to signal event status to a process, with little or no overhead. In order to use common event flags for process activation cueing, interrupt cueing, process deletion cueing, etc., association with a common event flag cluster is necessary. The $ASCEFC system service routine associates with one of four 32-flag clusters available within VMS. Only three of the clusters should be used. Use of cluster 0 is often self-destructive due to common use of these flags by individual routines so I recommend against the use of this cluster. Cluster 32 contains event flags from flag 32 through flag 63, cluster 64 contains flag 64 through flag 95 and cluster 96 contains flag 96 through flag 127. TELDAX uses cluster 64.

The three most important activities in real-time processing preparation, after parallel process creation, are-disabling the swapping of the process working set, locking working buffers in memory, and locking the working buffers to the end of the working set. These three actions guarantee that data and program logic will never be out-swapped and that the only context switching that occurs will be when output data is written to a storage device.

Disabling of process working-set swapping is done using the $SETSWM system service routine. The only parameter used is a bit set parameter. One indicates no swap, and zero indicates swap.

Cyclic multiple buffering of data being input to a real-time telemetry system is a foregone conclusion. Any type of processing and output of data to storage takes time and data input to a system is usually continuous data not burst data. Cyclic multiple buffering, as handled by the Sangamo-Weston Model 760 Buffered Data Channel, gives the user up to four buffer cycles of time to accomplish this processing and storage before the data is cyclically over-written. In order for cyclic multiple buffering to be effective the buffers must be locked in memory, or the time gained by multiple buffering will be more than lost by in-swapping of buffer areas.

Expansion of the working set region to allow for multiple buffer memory space is done by using the $EXPREG system service routine. By using the $LCKPAG system service the buffer region which has been added is locked in memory, and use of the $LKWSET system routine locks the entire working set in memory. These three system service routines guarantee no swapping of any necessary data or program logic.

The process is now ready to be activated, unless special hardware related initialization needs to be handled prior to real-time data acquisition.

EVENT FLAG COMMUNICATION

Process activation and deletion can be efficiently handled through the use of common event flags or local event flags. The more time-efficient choice is that of local event flags, but, for TELDAX needs, the limitations outweigh the slight gain in efficiency. TELDAX uses a separate event flag for the signalling of process start and process halt of each process being run in parallel. There are cases when more than one process is to be started at the same time. By using the same start event flag number for all processes being started simultaneously TELDAX is able to guarantee that this specification is met.

There are a number of ways that a user can initiate event flag usage. The system service routines $SETEF and $CLREF will set and clear specific event flags. The usage of $SETEF and $CLREF is the software method of flag toggling. Flags can also be set by associating the flag with a hardware interrupt device. The association of event flags to hardware interrupt devices is far more efficient than software toggling, when handled correctly, by sending the interrupt directly to the event flag with a $QIO service call. TELDAX uses the following procedure to implement this method. A channel is assigned to each hardware interrupt line used on the hardware interrupt device using the $ASSGN system service. For TELDAX purposes, the Sangamo-Weston Model 2765 Priority Interrupt Module works efficiently as it contains eight interrupt lines and is built to sit on a DEC Unibus (as does the Model 760 Buffered Data Channel). The asynchronous $QIO system service routine then connects the desired event flag with the hardware channel number using a set hardware code to set attention for when the interrupt is raised. In this way there is no separate software logic needed to set the event flag.

Once the event flags have been set up, the process waits for activation, deletion or other flag action, by invoking the $WAITER system service which waits for the raising of the event flag specified. When the event flag is raised, the operation indicated begins. In the case or the process activation this signals the beginning of real-time data acquisition or processing.

DATA COMMUNICATION

Kernel mode processing, as discussed in an earlier area of this paper, is the preferred mode of operation in real-time processing. Thus the use of RMS calls are not recommended, as they run in executive mode. For this reason all reading of input data and writing of output data to storage should be done using the $QIO system service routine. The availability of a built-in Asynchronous System Trap (AST) call directly from the VMS $QIO routine is a great advantage to the real-time user. Often, processing of data is done on a per record, per frame or per block basis. By reading data into the process in the size that is to be processed and stored, the completed $QIO routine can automatically cue an AST routine which is your processing and storage executive. This is very efficient for it allows for parallel processing and storage while cyclic buffering of input data is occurring. It should be noted that this is only efficient if the data is processed at a rate less than or equal to the rate of data input. If not, sooner or later the user will miss a block-end interrupt and will begin to lose buffers of data due to buffer overwriting.

GLOBAL COMMON AREAS

There is usually a need in data acquisition and number manipulation and monitoring systems for a data base of parameters which must be accessible to all processing streams. In TELDAX this is the case, and a global common area was chosen as the method for storing this data. By using a global common area, the parameter data base is linked to all processes and the main program. This allows for communication of data and local flags to all processes. A global common area is an installed region in memory which is accessible to all programs linked to the common area and all routines which refer to it using the proper COMMON statement in FORTRAN or .GLOBL statement in MACRO. For ease of routine reference, the global common area in TELDAX is in the form of a FORTRAN INCLUDE file.

This INCLUDE file is installed as the global common area and is also a part of every routine referring to a piece of data in the common area. A global common area is built as a block data program.

This block data program is compiled and linked and the executable code is INSTALLed in the system. All programs using this global common area must then be linked to this ".EXE" file using an OPT reference in the LINK program.

When deciding on whether or not to use a global common area, the advantages must be weighed against the disadvantages. The greatest disadvantage is that the data stored in the global common area is dynamic and at the will of both the users of the system and the system itself. Any program linked to the global common area can change that data by

writing over the current data with new data. Using mapped sections is one solution to this part of the problem. The data stored in the global common area is deleted every time the system is rebooted. This means that every time the VAX goes down the global common area is deleted and must be reinstalled with the data being recreated from scratch. Indirect command files called by the SYSTARTUP or LOGIN files will handle reinstallation, but data recovery must be handled on an application basis. The advantages to global common areas include the ability to communicate information on a global basis, dynamic updating of information stored in the common area, space efficiency and ease of installation. Each user must weight the advantages and disadvantages as they apply to the individual application being considered, before deciding for or against the global common area concept.

MAILBOX COMMUNICATION

Use of the mailbox utility of VMS was mentioned briefly in the discussion of the created process. The usage discussed in that area concerned pre-real-time activity. Mailboxing, in general, is not time efficient and involves a good deal of overhead. It is not highly recommended in real-time processing, but there are times when it is worthwhile. In TELDAX, we use a separate process which does nothing but handle the logging of errors occurring during real-time processing. The error logger double buffers the errors to guarantee that no loss of error data will occur while a buffer of error data is being written to the disk. The error information is passed, via the mailbox, by each process running, and it is possible for more than one process to be reporting an error at the same time if data rates are too high. This possibility of simultaneous reporting of errors, rules out the use of a global common area, as the loss of error data is possible. For this reason, error logging data is communicated via the error logging mailbox.

PROCESS DELETION

Deletion of created processes is handled internally to VMS in most cases. When a process is no longer active, a number of activities must take place. All open channels, files and connections must be closed.

This is done by using system services such as $DASSGN to reverse all $ASSGN’s, $DALLOC to reverse all $ALLOC’s, and $CANCEL to reverse all $QIO connections. The $SETSWM process swap mode routine is again called to enable swapping. All locked buffers and expanded regions are deleted using the $DELTVA delete virtual address system service. Finally, after any miscellaneous cleanup is completed, the process must exit. In most cases the FORTRAN END statement will accomplish this task. What actually occurs, and is used when a MACRO process is to exit, is the system invoking of the $EXIT and $DELPRC delete process system service microcode. This action ends the life

of the created process. If the deletion phase of the process life cycle is time critical to the rest of the system the user should beware. The $CANCEL and $DELPRC service routines can be slow enough to notice. The recommended solution is hibernation to guarantee that the process is deleted when the system expects it to be deleted.

SUMMARY

This paper has outlined an efficient and acceptable real-time telemetry processing environment and the ability to operate efficiently within the telemetry environment. The life cycle and communications methods discussed in this paper are used successfully in many applications on the VAX both with and without pre-processors.

In summary, this paper represents a philosophy of real-time telemetry processing which is more general than the VAX environment. It is a philosophy of time and space efficiency. It is the use of this philosophy on the VAX, with the resources which VMS offers and the real-time hardware offered by the industry, that make the marriage compatible and long-lasting.

This marriage has made TELDAX an efficient telemetry data acquisition, processing and storage system. It is user friendly and user expandable. Most importantly TELDAX is a reliable software system running on a reliable, efficient computer with a reliable and efficient set of telemetry front-end hardware.

BIBLIOGRAPHY

Computer Programming and Architecture, The VAX-11, Levy, Henry M. and Eckhouse, Richard H., Jr., Digital Press, Bedford, Massachusetts

VAX/VMS Volume 4 - I/O User’s Guide, Version V02, No. AA-D028B-TE, Digital Equipment Corporation, Maynard, Massachusetts

VAX/VMS - Internals and Data Structures, Version V2.2, No. AA-K785A-TE, Digital Equipment Corporation, Maynard, Massachusetts

VAX/VMS Volume 10 - System Manager’s Guide, Version V02, No. AA-D027B-TE, Digital Equipment Corporation, Maynard, Massachusetts

VAX/VMS Volume 4 - System Services Reference Manual, Version V02, No. AA-D018B-TE, Digital Equipment Corporation, Maynard, Massachusetts