Handling CAMAC Interrupts in Alpha OpenVMS/PCI

K.S. Lee, S.G. Kadantsev, E. Klassen, M.M. Mouat, P.W. Wilmshurst
Tri-University Meson Facility (TRIUMF)
4004 Wesbrook Mall, Vancouver BC V6T 2A3, Canada

Abstract

Software for Alpha/OpenVMS systems has been developed to support CAMAC interrupts (LAMs) via PCI bus. A number of devices in TRIUMF's Central Control System generate interrupts that are delivered via CAMAC systems. These interrupts arrive using previously existing CAMAC executive crates and system crate interfaces. Until this development, these interrupts were only being serviced by VAX/OpenVMS computers using Qbus but the tendency to replace VAXes by Alphas has required that this LAM handling software be developed. The initial requirements, hardware and software configuration, driver structure, and performance are described.

1 Introduction

TRIUMF's Central Control System consists of a mixture of VAXes and Alphas running OpenVMS. These computers are organized into two mixed OpenVMS clusters, a production and a development cluster. Access to cyclotron devices occurs through a CAMAC executive crate architecture. Each executive crate supports up to seven serial and/or parallel CAMAC branches and allows multiple CPUs access via hardware arbitration. TRIUMF Controls group supports 5 executive crates. Each node in each cluster is capable of addressing two or more of these executive crates. The CAMAC operations in both types of CPU are provided by a set of user-written system service routines in the form of an installed, privileged shareable image. This method was chosen over using a formal CAMAC device driver for performance reasons, the reduced overhead involved in servicing the CAMAC operations.

A variety of devices in TRIUMF's Central Control System such as the main operator console, the remote console panels and protect monitors, generate interrupts (LAMs) that are delivered via the CAMAC systems. The CAMAC interrupts on the VAX/OpenVMS computers arrived via the Qbus are handled using the VMS connect-to-interrupt driver (CONINTERR.EXE). When the interrupt occurs, the CONINTERR will cause an asynchronous trap (AST) procedure to execute in process context. A nonprivileged process called LAM_HANDLER in the VAX gets an IO channel for the CAMAC device GEC0 and issues a queued IO system call ($QIO) with the IO$_CONINTREAD function code specifying an AST procedure to execute and an event flag to be set when the interrupt is generated. There is a tendency is to replace the existing VAXes with Alpha machines due to the better price/performance of the Alphas. Unfortunately the connect-to-interrupt driver that is available under VAX/OpenVMS is not available under Alpha/OpenVMS. This meant that migrating LAM related applications from the VAXes to the Alphas was not immediately possible.. As a result, a formal CAMAC device driver was developed for handling the interrupts delivered via the CAMAC systems.

2 Hardware configuration

Two Alphas, an AlphaServer 2000 5/250 and an AlphaStation 600 5/266, each with a PCI local bus, are connected to the CAMAC system via the Logical Company BCI-2100 Q-bus adapter[1]. A pair of TRIUMF designed executive crate interfaces (0782/0783 modules) form the connection between the Q-bus and the GEC executive crate (figure 1). This architecture allows multiple Alphas to access multiple executive crates and is found to be quite flexible.

The PCI Q-bus adapter consists of a BCI-2100 PCI controller, a CAQ-2101 Q-bus module, and a CAB-1104-8 interconnect cable. The PCI Q-bus adapter enables the computer to read and write to the Q-bus address space and to control Q-bus interrupt requests and DMA transfers. Currently, CAMAC interrupts are only generated in one executive crate (GEC0).

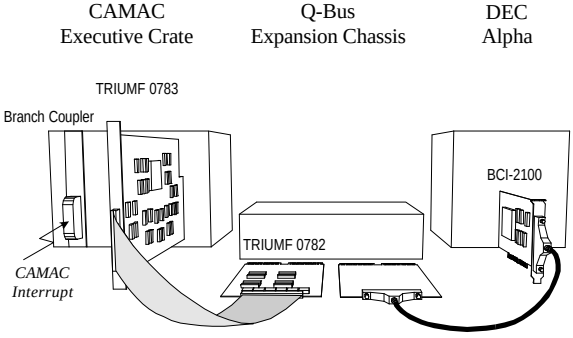

Figure 1. Hardware Setup for handling CAMAC Interrupt

3 Software configuration

A formal CAMAC device driver, written in C, was developed with the sole purpose of handling the CAMAC interrupts. The benefit of using C is that it is easier to write and allows a certain level of portability. Also, for RISC machines, a good optimizing C compiler generates machine code usually more efficient than codes using the MACRO-64 language. The driver does not support any other CAMAC operation such as read and write operations.

Those are supported by the existing user-written system service routines in the form of an installed, privileged shareable image.

The CAMAC driver offers a SETMODE function which requests an attention AST be delivered to an AST routine in process context. This driver code also contains an interrupt service routine which acknowledges a CAMAC interrupt has happened and delivers an attention AST.

The CAMAC device driver is loaded into system virtual address space with its associated data structure in nonpaged pool using the system management utility (SYSMAN) IO CONNECT command:

$ SYSMAN

$SYSMAN>IO CONNECT CMC0 /driver = cmc_driver /vector = 20 /node = 64/ adapter = 2 /csr = 0

A nonprivileged process called LAM_HANDLER gets a IO channel for the CAMAC device CMA0 and issues a $QIO call with the IO$_SETMODE function code specifying an attention AST procedure to execute after the interrupt service routine has completed. The AST routine performs the necessary CAMAC operation to fully service the interrupt and sets an event flag to trigger the re-arming of the AST notification with another $QIO call (figure 2).

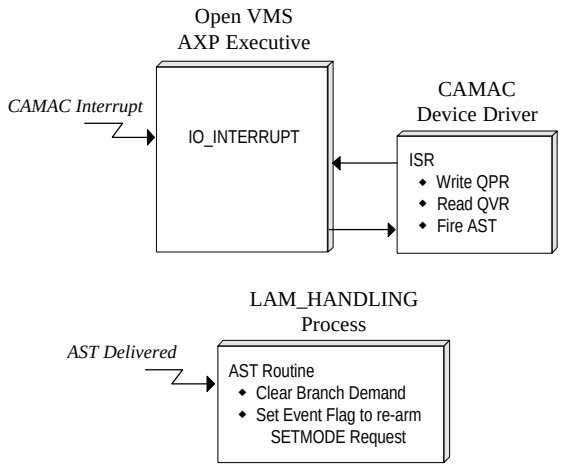

Figure 2. Control Flow for Handling CAMAC Interrupts

4 Driver structure

A device driver has to interact with the operating system to process the specified I/O request[2]. In this case, the CAMAC device driver has to process a SETMODE I/O request. The driver consists of a set of routines and data structures in nonpaged pool. The following is a brief description of the CAMAC driver routines.

4.1 Initialization routines

A set of initialization routines are executed when the CAMAC driver is first loaded. This includes the driver table initialization routine, the structure initialization routine, the structure reinitialization routine, the csr_mapping routine and the unit initialization routine.

The driver table initialization routine initializes the Driver Prologue Table (DPT), the Driver Dispatch Table (DDT) and the Function Decision Table (FDT) structures.

The structure initialization routine initializes the device Interrupt Priority Level (IPL) to 21 and the fork lock index to 8. It also specifies the CAMAC device characteristics.

The structure reinitialization routine sets up the Interrupt Transfer Vector Block (VEC) of the Channel Request Block (CRB) to point to the interrupt service routine. It initializes the pointers from the Device Data Block (DDB) to the Driver Dispatch Table (DDT) and the Unit Control Block (UCB).

The csr_mapping routine maps the PCI configuration space, the PCI memory space and I/O space into the Alpha's virtual address space by calling the platform independent I/O bus mapping routine IOC$MAP_IO.

The unit initialization routine enables interrupt by writing appropriate values to the Q-bus Control and Status Register (CSR) and the Q-bus Priority Register (QPR).

4.2 FDT routine

The CAMAC driver has only one FDT routine for handling the set mode function. It enables attention ASTs so any process issuing a $QIO call with this function will have an attention AST delivered when the CAMAC device interrupts. It calls the COM_STD$SETATTNAST to do the following: it allocates an expanded access control block (ACB) from non-paged pool to hold the AST procedure value, AST argument, channel number and PID. It then acquires the device lock, raising IPL to 21, to synchronize access to the attention AST list[3]. After inserting the ACB into the attention AST list it releases the device lock and restores the previous IPL.

4.3 Interrupt service routine

This routine is executed when a CAMAC interrupt happens. It executes at device IPL 21 and calls COM_STD$DELATTNAST to delivers all queued attention ASTs. It enables further processing of interrupts by writing to the Q-bus Priority Register and reading the Q-bus Vector Register (figure 3). In an environment where multiple executive crates can deliver interrupts, the content of Q-bus Vector Register can be examined to find out which executive crate has generated the interrupt.

5 Performance

Timing results are shown in table 1. The interrupt handling time is measured as the length of time that the branch demand is present. The VAX 4100 has the slowest CAMAC interrupt handling time followed by the VAX 4105. The Alphas are quite a bit faster. The interrupt handling time continues to decrease as the CPUs get faster.

Handling time gradually increases with an increase in

system load. This could be explained by the increased

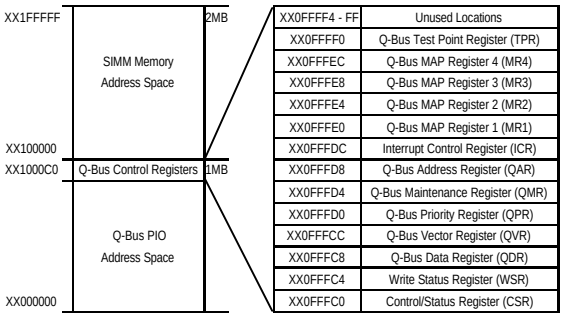

Figure 3. Q-bus Control Address Space

System	Length of branch demand
AlphaStation 600 5/266	150 µsec
AlphaServer 2000 5/250	184 µsec
VAX 4105	233 µsec
VAX 4100	384 µsec

Table 1: Timespan of branch demand present when system is 1-3% busy

System Business Percentage	AlphaStation 600 5/266	AlphaServer 2000 5/250
~ 1 %	150 µsec	184 µsec
~ 10 %	166 µsec	202 µsec
~ 20 %	173 µsec	211 µsec
~ 30 %	183 µsec	212 µsec
~ 50 %	184 µsec	227 µsec
~ 80 %	195 µsec	230 µsec

Table 2: Timespan of branch demand present with varying system load

activities at IPL (interrupt priority level) above zero which will lengthen the CAMAC interrupt handling time (which happens at IPL 23) as well as the attention AST delivery time (which happens at IPL 2).

As expected, there is also a dependency on the system load. Table 2 shows that t he average CAMAC interrupt

The nature of the system load also affects the timing. As we see in table 3, when the Alpha is kept 100% busy doing only CAMAC cycles, the increase in the CAMAC interrupt handling time is very small. Whereas there is a greater increase in time when the computer is doing various activities such as updating X-terminal displays, accessing disks, the network and other peripherals.

System	Length of branch demand
AlphaStation 600 5/266	153 µsec
AlphaServer 2000 5/250	186 µsec

Table 3: Timespan of branch demand present with system kept busy at 100% doing CAMAC cycles

The results indicate at least a one-third improvement in CAMAC interrupt handling time from a VAX 4105 to an AlphaStation 600.

6 Summary

To facilitate the migration of programs from VAXes to Alphas, a CAMAC device driver was developed in the Alphas for handling interrupts delivered via the CAMAC systems. The hardware and software structure of the device driver is discussed and the interrupt handling timing results are presented.

In general, the interrupt handling time in the Alphas is faster than that in the VAXes. A dependency on the CPU power and the system load is observed. The current driver supports CAMAC interrupts generated in a single executive crate (GEC0). However, it is possible to support interrupts from multiple executive crates by examining the Q-bus Vector Register.

References

[1] BCI 2100 Q-bus Adapter for the PCI Local Bus Owner's Manual (The Logical Company)

[2] Writing OpenVMS Alpha Device Drivers in C by Sherlock & Szubowics (Digital Press)

[3] OpenVMS AXP Internals and Data Structures by Goldenberg & Saravanan (Digital Press)