A WINDOWS NT DEVICE DRIVER FOR THE PCI TO CAMAC EXECUTIVE CRATE INTERFACE

K. S. Lee & P. J. Yogendran,
TRIUMF, Vancouver, Canada

Abstract

This paper describes the development and present status of a Windows NT device driver for the TRIUMF PCI to CAMAC interface. The TRIUMF 500 MeV Cyclotron Central Control System currently supports the PCI to CAMAC interface using OpenVMS for device access. A project to investigate device access using Windows NT is underway. The first step in this investigation has been the development of a Windows NT device driver. Subsequent work will be necessary to port existing device access software. In the paper, the software design, implementation, and performance of the device driver is discussed.

1 INTRODUCTION

A PCI to CAMAC interface has been developed for the TRIUMF's Central Control System to provide access to its multiple executive crate CAMAC system for ALPHA machines running OpenVMS[1]. Currently the PCI bus is widely supported on many different platforms including Intel, Macintosh, PowerPC and ALPHA. A project was embarked to explore the PCI to CAMAC interface in a PC running the Windows NT V4.0.

2 SOFTWARE DESIGN

Since the Windows NT system does not allow user-mode code to directly access I/O hardware device, a kernel mode driver is written to provide CAMAC access. It is a collection of routines interacting with the Windows NT system to provide access to one or more PCI to CAMAC interface cards. It is a monolithic driver for it does not need to cooperate with other drivers to achieve the IO access. It deals with many data structures containing device information. They are allocated in the system non-paged memory pool and are referred to as objects in the Windows NT environment [2].

A CAMAC driver object is created when the I/O manager loads the CAMAC driver (figure 1). At this time, the PCI bus is probed and one or more PCI to CAMAC interface card may be detected.

An interface card is capable of having four executive crates attached to it. A unique device name is given for each executive crate found attached to the card. Each device is represented by a device object and a device extension object (figure 2). The device object contains a standard set of data expected by the I/O Manager and the device extension contains specific device information such as the device status, hardware port address and pointer to the I/O buffer.

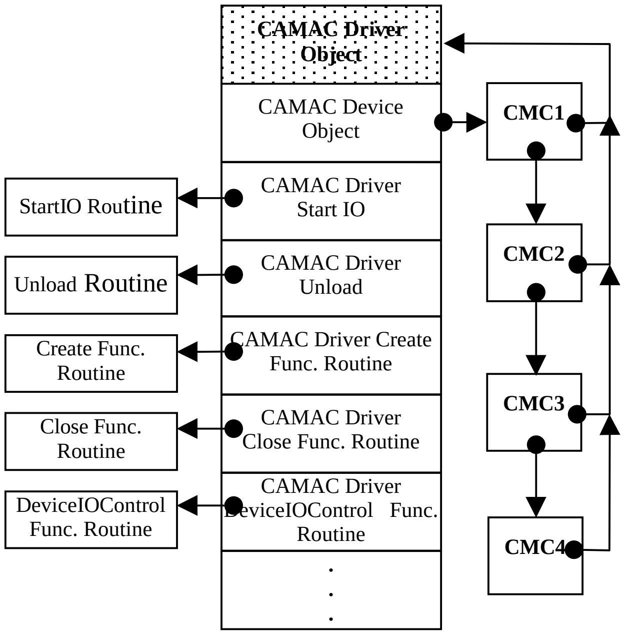

Figure 1 Structure of the CAMAC Driver Object

Due to internal register sharing, only one CAMAC cycle can happen at one time in a card. To synchronize device access, a CAMAC controller object and a controller extension are introduced (figure 3). A CAMAC device can only start its operation when it is granted ownership of the CAMAC controller object. Upon finishing its operation, the CAMAC device would release the CAMAC controller object so another operation could go ahead. The CAMAC controller extensions holds information that is common to all devices such as common hardware port address.

When a user-mode application performs an I/O operation to the CAMAC system, it initiates a CAMAC I/O request. The I/O Manager constructs a single I/O Request Packet (IRP) as a result[3]. It passes the IRP to the CAMAC driver dispatch routine. After checking the validity of the request, the CAMAC start I/O routine is called.

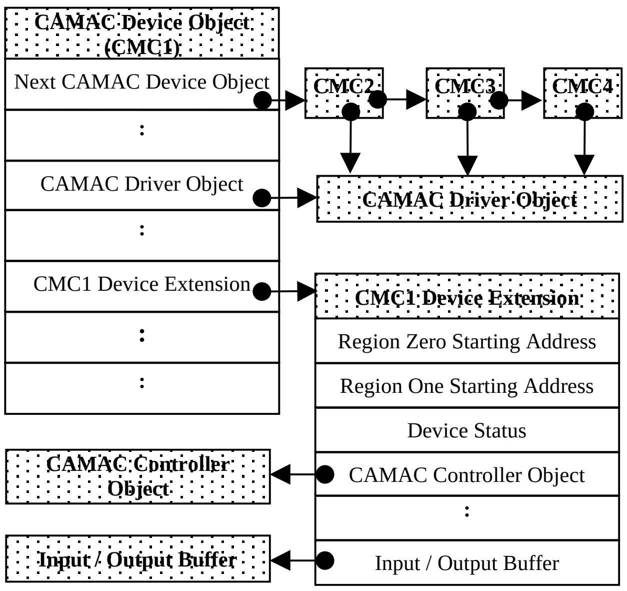

Figure 2 Structure of a CAMAC Device and Device Extension object

The interface card handles all CAMAC operations as programmed I/O. When a CAMAC cycle is finished, the DONE bit in the POL register is set. As a result, a polling method is used in the start I/O routine to detect the change in status. Since a CAMAC cycle typically finishes in under 10 microseconds, this would not pose serious performance problems for the Windows NT system. When the operation is finished, the IRP is released back to the I/O Manager and the application will be notified about its completion.

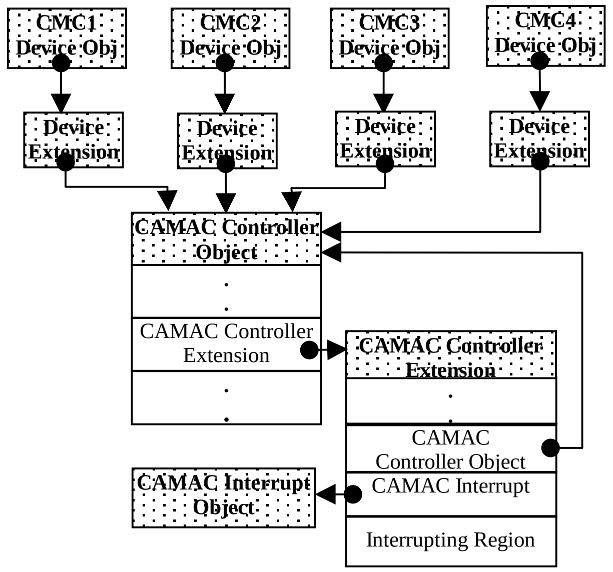

Figure 3 Structure of a CAMAC Controller and Controller Extension object

3 IMPLEMENTION

The CAMAC device driver entry points are discussed in the following:

3.1 CAMAC Driver Entry Routine

In this routine the driver queries the PCI bus to locate one or more CAMAC interface cards by calling the HalGetBusData() routine. This routine belongs to a set of system services in Windows NT called Hardware Abstraction Layer (HAL) which interacts with the processor resources and hardware. This layer makes developing a single device driver on multiple platforms much easier. Once the driver finds the card, HalAssignSlotResources() is called to allocate resources for the interface card. The purpose is to prevent other drivers from touching the same hardware.

The routine also initializes the driver object by declaring other driver entry points such as StartIO routine, Unload routine and Major Function routines. Since it is necessary to synchronize CAMAC access to different executive crates, IoCreateController() is called to create a CAMAC controller object and its Extension for each interface card found. Four CAMAC devices are created for every card by calling IoCreateDevice() to provide access for the four possible executive crates.

The routine also initializes the interface card to a known state. Since this involves accessing some internal registers in the card, the KeSynchronizeExecution() is called to run the code at Device Interrupt Request level (DIRQL) to prevent any conflicts between the code and the Interrupt Service routine.

3.2 CAMAC Dispatch Routine

The routine performs data sanity checks before allowing the IRP to go further. This device driver supports the dispatching of four major functions to the CAMAC device: Create, Close, Device Control and Read. The Create function is mandatory in Windows NT. The Close function provides CloseHandle() call for the device. These two functions do not require actual device operations. So upon their completion, the IRP is released back to the I/O Manager.

The Device IO Control and the Read functions involve actual device operation and data transfer. The Device Control function handles reading and writing to the internal registers as well as executing the full set of CAMAC cycles. There are five I/O control codes corresponding to five different types of operations. The discussion of the hardware aspect of the interface card can be found in another paper presented in this conference[3]. The data transfer method is buffered IO. The Read function deals with in-coming unsolicited CAMAC interrupts. Upon completion of these two functions, the IRP is passed to the Start I/O Routine.

3.3 CAMAC Start I/O Routine

For the Device IO Control function, the Start IO routine calls IoAllocateController() to gain ownership of the controller object before performing any actual device operation. The code that accesses the hardware is done within a KeSynchronizeExecution(). routine to prevent conflict with the interrupt service routine. As discussed earlier, a polling mechanism is used for the programmed I/O data transfer. This is done by calling KeStallExecutionProcessor(). If any errors occur during the I/O operation, the proper error code is set and the operation is terminated. Regardless whether the operation is successful, IoFreeController() is called to release the controller object and the IRP is released back to the I/O Manager.

3.4 CAMAC Unload Routine

This routine is called by the I/O Manager when a user requests to stop the driver from the Control Panel's Device applet. First the interrupts are disabled from the CAMAC device and IoDisconnectInterrupt() is called to remove the CAMAC interrupt object.

For each CAMAC device, IoDeleteSymbolicLink() and IoDeleteDevice() are called to have the device removed from the system. After all the devices are removed, IoDeleteController() is called to remove the CAMAC controller object. Last, the driver-wide resource is deallocated by calling IoReportResourceUsage().

4 PERFORMANCE

CAMAC cycle timing data has been collected in three models of PC running Windows NT V4.0. The result is shown in Table 1. A CAMAC cycle time includes both the hardware I/O access time as well as the software setup time. The software time is expected to scale with the CPU power. This is confirmed by the results.

A Dynamic Link Library (DLL) has been written to support standard IEEE calls such as CDREG and CFSA calls. This will facilitate the future porting of device access applications to the Windows NT

Table 1: CAMAC Cycle Timing

PC model	Read Cycle	Write Cycle
166Mhz Pentium	59 usec	56 usec
350Mhz Pentium II	26 usec	23 usec
450Mhz Pentium III	21 usec	20 usec

5 SUMMARY

A kernel mode device driver has been developed for the PCI to CAMAC interface card to run under Windows NT V4.0. The software design and implementation are discussed and timing data for CAMAC cycle is collected for different models of PCs. The results show performance is acceptable and scales with CPU power. A DLL image was developed to provide CFSA and CDREG calls. This is the first step in exploring device access in the Windows NT system.

This CAMAC device driver is developed under Windows NT V4.0 and can run under Windows NT V5.0 (also known as Windows 2000) without changes. However, modifications are needed for it to incorporate new functionality such as plug and play and power-management. Notable changes are the ones in Driver Entry Routine and in the handling of new type of IRPs.

REFERENCES

[1] P. Wilmshurst & K. S. Lee, "PCI To CAMAC Executive Crate Interface", ICALEPCS 99.
[2] A. Baker, "The Windows NT Device Driver Book: A Guide for Programmers "
[3] P. G. Viscarola & W. A. Mason, "Windows NT Device Driver Development"