User Guide of Pipeline CAMAC Controller with PC104plus Single Board Computer (CC/NET)

Yoshiji YASU, Eiji INOUE,
Shuichi HARADA(1) and Haruyuki KYOO(2)

Online group, Institute of Particle and Nuclear Studies
High Energy Accelerator Research Organization(KEK)
(1) TOYO Corporation
(2) Fird Corporation

November 2003

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview of Pipeline CAMAC Controller

CAMAC[1] is a IEEE standard electronics and still used in High Energy and Nuclear Physics experiments. Lots of CAMAC controller were developed so far, but the pipeline CAMAC controller is an epoch-making controller because the throughput achieves approximately up to 3 MB/s with 24-bit data. The controller occupies 2 slots(station 24,25) in a CAMAC crate. The controller is located in the crate shown in Figure 1.1. It consists of a PC104Plus-based single board computer, PCI interface (PCI control unit) and CAMAC interface (CAMAC control unit). The PC104Plus is a standard PCI specification for embedded systems[2] and the low-power board computer includes a flash disk with IDE interface, Fast Ethernet, USB and so on. The PCI and CAMAC interfaces consist of a ALTERA FPGA[3], respectively. The controller also has a DAQ function for event numbering. The signal handling of trigger-input and busy-out are implemented while the event count can be read out.

Linux system[4, 5] can run on the pipeline CAMAC controller while the device driver and the library[6, 7, 8] are also provided.

The pipeline CAMAC controller and the software are developed by KEK[9], TOYO[10] and Fird[11].

1.1.1 Functionality

There are a CAMAC Executor, a CAMAC LAM Handler (CAMAC Interrupter), a DAQ Executor and a DAQ Trigger Handler (DAQ Interrupter) in the pipeline CAMAC controller. The Executors accept a packet including at least a command frame or more from CPU via PCI, execute them and then send reply frames to CPU. The Handlers generate a packet including a reply frame and then send to CPU. When LAM or Trigger interrupt occurs during the executors processes a packet, the generation of interrupt reply frame will be postponed. After the packet is processed, the interrupt reply frame will be sent.

The pipeline CAMAC controller can execute CAMAC command with small overhead continuously by receiving the command frames from a computer memory and sending the reply frames to the computer memory. The controller also includes a DAQ function for event numbering. The NIM signal of trigger input and busy out can be handled while the event count can be read out.

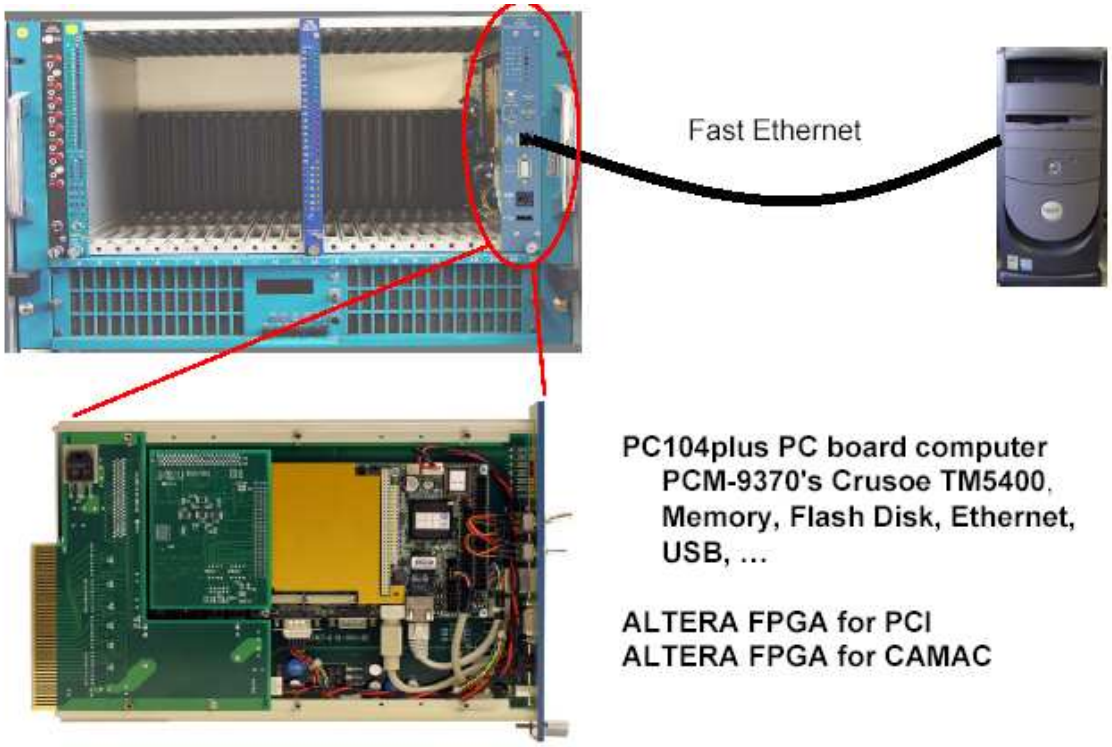

Figure 1.1: Configuration of Pipeline CAMAC Controller

The size of command and reply frames is 64-bit. It consists of 8-bit header and 56-bit payload. The payload for basic CAMAC command frame contains CAMAC station number N, CAMAC sub-address A, CAMAC function F and data to be written if necessary. That for the reply frame additionally contains CAMAC status such as Q & X and read data if the CAMAC function is 'read'. That for CAMAC LAM includes 24-bit LAM pattern and that for DAQ Trigger includes 32-bit event count. That for DAQ Executor can clear busy-out signal while the signal disables next event trigger.

1.1.2 Performance

The pipeline CAMAC controller can execute a CAMAC access in 1 usec + small overhead (40 nsec), that is, the throughput is approximately 3 MB/sec with 24-bit data. Figure 1.2 shows the performance of the controller. An unit of mesh in the Figure is 1 usec. CAMAC Busy and CAMAC S1 signals are shown.

1.1.3 Pipeline Method:Why so fast?

From historical point of view, CPU speed of PC and the Bus speed are getting faster and faster. However, the actual CAMAC access speed of the controllers so far is slow and far from maximum CAMAC speed, 3 MB/sec. There is a reason why it is so slow. It is already proved that pipeline method is useful for speed up on DAQ system. Lots of architectures related to pipeline method were developed in 1990s. However, no CAMAC controller adopted the pipeline method so far. The key point is to adopt a true pipeline from CAMAC command in CPU memory to the result (e.g., readout data) in CPU memory.

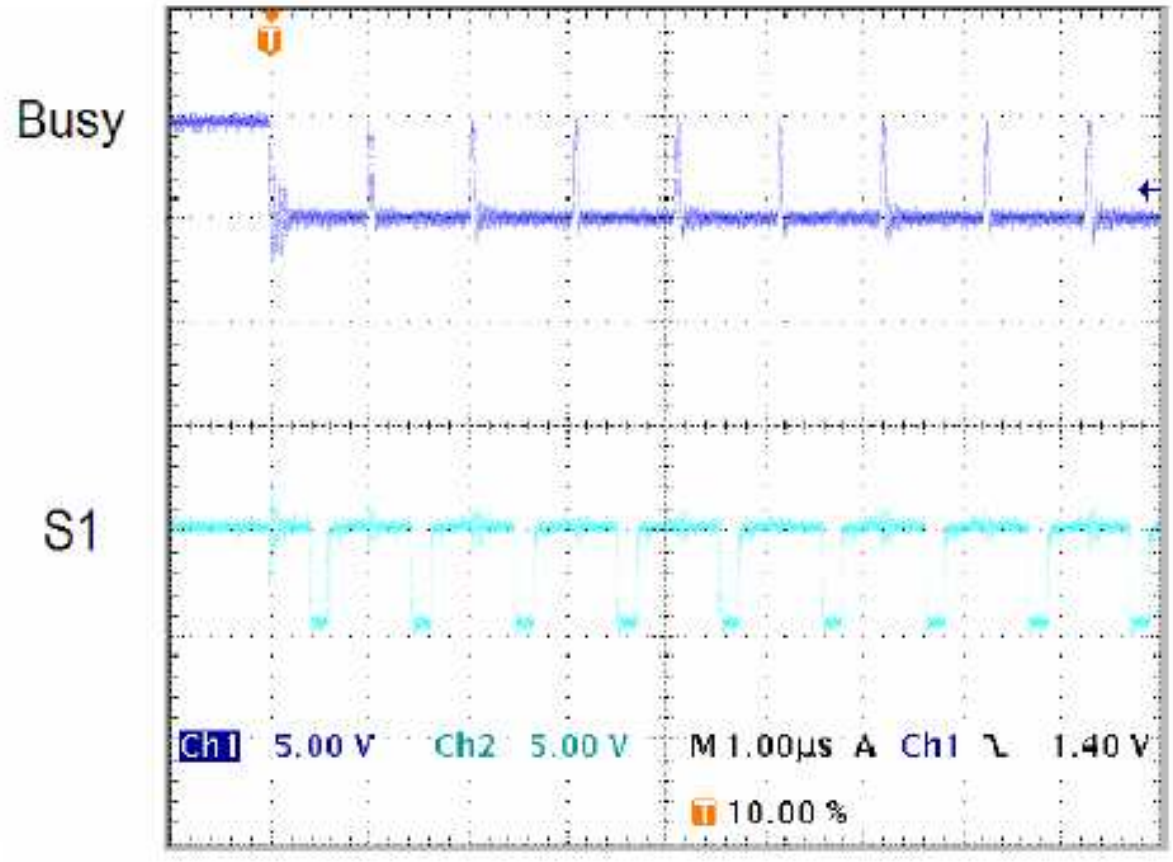

Figure 1.2: Performance of Pipeline CAMAC Controller

Some CAMAC controller partially adopted the pipeline method using a list processing[12], but it was not the true pipeline. Thus, the throughput did not achieve maximum CAMAC speed 3 MB/sec because the pipeline was destroyed at some point. The pipeline CAMAC controller realized the true pipeline.

The architecture of the pipeline CAMAC controller is completely different from that of old style of CAMAC controller so far. Thus, the operation of the pipeline CAMAC controller is also different from that of the old CAMAC controller. For the operation on the old CAMAC controller, N (station number), A (sub-address) and F (function) are put into an I/O register in the CAMAC controller and then the go bit of the register is set for starting the operation. Then, the status of the completion is checked and the data are read. After that, next operation of CAMAC command will start. The old CAMAC controller does not adopt the pipeline method. The pipeline CAMAC controller adopts the truly pipeline method. The pipeline method enabled the controller maximum speed of CAMAC, approximately up to 3 MB/sec with 24-bit data. On one hand, the pipeline CAMAC controller sends multiple command frames, which include N, A, F and data if necessary, via Tx I/O registers in the PCI control unit. On the other hand, the controller receives multiple reply frames, which also includes N, A, F, status(Q and X) and data if necessary, via Rx I/O registers in the PCI control unit. The operations to Tx and Rx are done concurrently. If DMA is used, the throughput reaches maximum speed. The Figure 1.2 is the timing of CAMAC Busy and CAMAC S1 signals when the DMA starts. DMA function is provided on old CAMAC controller, but the function is limited such as Address-scan, Q-stop and so on. For the pipeline CAMAC controller, any operation to CAMAC are executed in DMA mode. There is no limitation. From DMA point of view, the pipeline CAMAC controller is also completely different from old CAMAC controller.

1.2 Organization

The readers of this guide may not read all of chapters. This section introduces the contents of all chapters in the guide briefly. Thus, the readers can jump to the chapter they need.

This chapter 1 shows the overview of pipeline CAMAC controller. They should read this chapter first to understand the pipeline CAMAC controller.

The chapter 2 introduce the hardware components. The chapter describes the detail of three components, PC104-Plus board computer, PCI control unit and CAMAC control unit. There are also the descriptions of the front panel and the power consumption.

The chapter 3 describes not only the PCI operation but also the frame format of CAMAC/DAQ command/reply. The PCI registers and the bit assignment are shown. The special CAMAC functions are also explained. If the readers like to understand the detail of how to operate the registers, read the section of operation procedure.

The chapter 4 explains how to get the distribution kit of CAMAC device driver and the library, how to install the driver and the library, the usage and the examples. It describes the detail of calling sequence of the CAMAC libraries while there are a general purpose CAMAC library and a dedicated CAMAC library for the pipeline CAMAC controller.

The chapter 5 describes Linux system and the applications running on the pipeline CAMAC controller. The chapter introduces how to recover the Linux system although a Linux system tailored by KEK will be pre-installed into a compact flash disk at purchase of the pipeline CAMAC controller. Two application programs are shown. One is remote

CAMAC library and another is a WEB-based CAMAC executor. When the remote CAMAC library is used, a CAMAC program running on the pipeline CAMAC controller (the board computer) can run on remote computer without any modification. By using WEB-based CAMAC executor, a simple CAMAC operation can be done on a WEB navigator such as Internet Explorer.

The chapter 6 shows the results of the performance measurement. It includes Linux system performance measured by a benchmark program **lmbench**, **netperf** and so on. From the results from **lmbench**, Latency of operating system entry, process creation costs and context switching costs are shown. **netperf** provides TCP throughput performance and TCP response / request performance. Memory copy performance is measured by a simple program. The CAMAC performance is also included in the results. It consists of performance of basic functions, block transfer performance, interrupt frame performance and CAMAC performance with the software library. From application point of view, CAMAC remote access program is evaluated.

This user guide is available in the URL[13].

Chapter 2

Hardware Components

The pipeline CAMAC controller is located into 24th and 25th slots of CAMAC crate. It mainly consists of a PC104Plus Single Board Computer, a PCI control unit and a CAMAC control unit. Figure 2.1 shows the side view of pipeline CAMAC controller called CC/NET. There are four boards as the components. The left side board including CAMAC card edge connector in the front is called D board. It includes DC-DC converter and CAMAC line driver. The center board including gold plate in the front is called C board. It is the board computer. The smallest board in the front is called A board, which is connected to the C board via PC104Plus PCI bus[2]. It is the PCI control unit. The rear board is called B board, which is connected to the A board via a general purpose interconnect called CSP[14]. The CAMAC control unit consists of the D board and the B board.

2.1 PC104plus Single Board Computer

The board computer is Advantech PCM-9370[15], 3.5" (145 mm x 102 mm) Transmeta Crusoe 500 MHz processor Single Board Computer including TM5400 processor, 310 MB memory, two IDE UltraDMA 33 mode up to 33MB/sec, LCD/CRT controller, 10/100 Mbps Ethernet controller, two 1.1 compliant USB ports, mini-din connector for keyboard, PS/2 mouse and so on. The board computer also support an Embedded PCI protocol, PC/104-Plus.

2.2 PCI control unit

The A board includes a ALTERA FLEX10KE100 FPGA. On one hand, the board connects to CPU board via 4x30 (120-pin) 2mm pitch stack-through connector for PC104plus PCI. On the other hand, the board also connects to CAMAC control unit via 2x40 (80-pin) TX14 series connector for a general purpose interconnect called CSP[14].

2.2.1 Architecture

The block diagram of PC control unit is shown in Figure 2.2. There are, Tx/Rx FIFO, Tx/Rx PCI engines and PCI multiplexer for Tx/Rx. Tx and Rx PCI engines work independently while PCI multiplexer manages the PCI usage for Tx and Rx PCI engines.

Figure 2.1: pipeline CAMAC controller

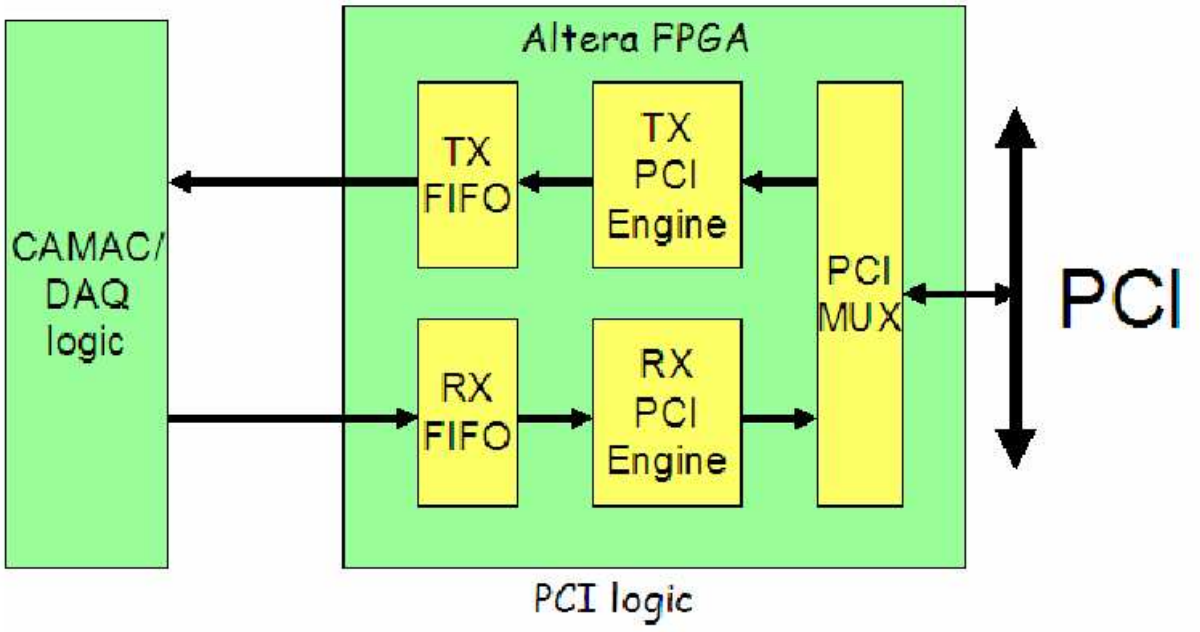

Figure 2.2: Block diagram of PCI control unit

2.2.2 Components: Tx/Rx FIFOs, Tx/Rx PCI engines and PCI multiplexer

Tx/Rx FIFOs have the depth of 256 with 32-bit width each. Tx/Rx PCI engines have two modes, master and slave. The slave means PCI slave which the PCI interface becomes at the operation of PCI registers from CPU. In master mode, the PCI interface can transfer data from/to memory in computer without CPU intervention. Thus, master mode not only reduces CPU usage but also realizes fast transfer in comparison with the slave mode.

The Tx PCI engine receives a least a command frame via PCI bus and then sends it to the Tx FIFO. The Tx FIFO sends the command frame to a CAMAC executor via the CSP. The Rx FIFO engines receives a least a reply frame via the CSP and then sends it to the Rx PCI engine. The Rx PCI engine send the reply frame to PCI. The PCI multiplexer shares the PCI usage with Tx/Rx PCI engines. Thus, Tx/Rx PCI engines can work independently.

2.3 CAMAC control unit

The unit in the Figure 2.3 includes the B board located into 25th CAMAC station and the D board located into 24th CAMAC station. The B board is a main board of CAMAC control unit, which contains ALTERA FLEX10KE50 FPGA. The B board is connected to the D board via 2x40 (80-pin) TX14 series connector. The D board contains DC-DC converter for the C board and the line driver for CAMAC bus.

The FPGA includes basic CAMAC execution by CAMAC Executor, CAMAC LAM handling by CAMAC Interrupter, execution of DAQ functions by DAQ Executor and DAQ trigger handling by DAQ Interrupter as the functions.

2.3.1 CAMAC Executor

The executor receives CAMAC command frames from the PCI control unit and then executes CAMAC cycles. It sends CAMAC reply frames as the result to the PCI control unit via the packet control circuit.

2.3.2 CAMAC Interrupter

The interrupter generates a LAM interrupt frame as CAMAC reply frame when an interrupt occurs on a CAMAC module. The reply frame is sent to the PCI control unit via the packet control circuit.

2.3.3 DAQ Executor

The executor receives DAQ command frames from the PCI control unit and then executes the DAQ functions. It sends the result frames as DAQ reply frames to the PCI control unit via the packet control circuit.

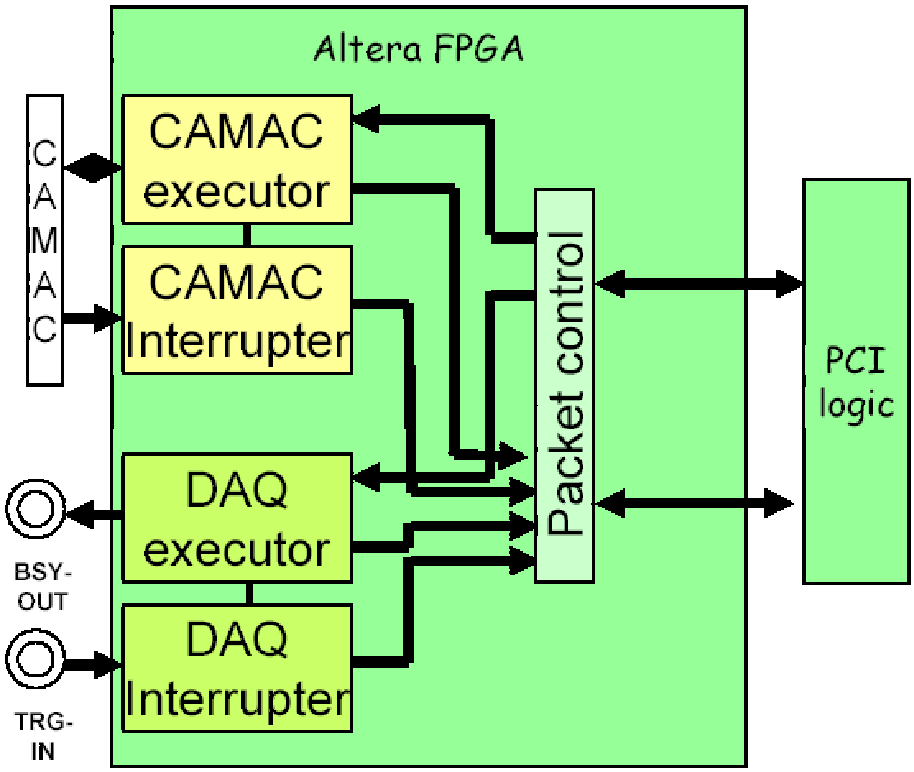

Figure 2.3: Block diagram of CAMAC control unit

2.3.4 DAQ Interrupter

The interrupter generates a DAQ trigger interrupt frame as DAQ reply frame when a DAQ trigger is input from TRIG-IN LEMO connector. The reply frame is sent to REPLY FIFO via the packet control circuit.

2.4 CSP: a general purpose interconnect between PCI and CAMAC

The PCI control unit and CAMAC control unit are connected by the interconnect. The interconnect is designed in general. It can handle simple packet transfer. A packet contains multiple frames. For an example, the frame size for CAMAC is 8 bytes. This means 4 cycles of the interconnect is necessary for a CAMAC frame.

2.4.1 Introduction

When ReadyToWrite and ReadyToRead signals are asserted, data can be sent and received. A packet transfer begins at this moment. The transmitter should write data and the receiver should read it at next clock when EnableToSendPacket and EnableToReceivePacket signals are both asserted. When ReadyToWrite or ReadyToRead signal is deasserted, a packet transfer should be terminated. When EnableToWrite or EnableToRead signal is deasserted, a packet transfer should be paused.

Table 2.1: CSP signals

I/O Signal name	Description
ReadyToWrite	Ready to write a packet or start writing a packet
EnableToWrite	Enable to write data in a packet
ReadyToRead	Ready to read or start reading a packet
EnableToRead	Enable to read data in a packet
Clock	Up to 25MHz
Data	16bits
Data width	1 means 16-bit width and 0 means 8-bit width
Tx Information	4 bits for transmitter
Rx Information	4 bits for receiver

2.5 Front Panel

Figure 2.4 shows a front panel of the Pipeline CAMAC controller.

2.5.1 LEDs

There are several LEDs. The CPU LED is used for indicating status of CAMAC control unit. If the LED blinks, CAMAC control unit is not ready. Thus, a flag SYS_READY in System register should be set. The BUSY LED is direct reflection of CAMAC Busy signal. The NO-X and NO-Q LEDs mean final states of CAMAC X and CAMAC Q, respectively. If CAMAC X or CAMAC Q is not set, the NO-X or NO-Q is light. The L-SUM LED is light when one of CAMAC modules issues LAM and the LAM is enabled. The I LED means final state of CAMAC Inhibit. The IE LED is light when the interrupt of the controller is enabled.

2.5.2 Switches

There are two toggle switches. One is for setting CAMAC Online and Offline. Another is for generating CAMAC Z and CAMAC C.

There is a small push-button for reseting the Pipeline CAMAC controller. The reset makes the power down/up.

2.5.3 Trigger-Input and Busy-Out

Two LEMO connectors of NIM signals for Trigger Input and Busy Out are used for event numbering of DAQ.

2.5.4 Ethernet, USB, VGA and PS/2

There are Fast Ethernet connector, VGA connector, PS/2 connector and USB connector. The USB is master device, not target device. Thus, the pipeline CAMAC controller can not be handled via the USB.

Figure 2.4: Front Panel of Pipeline CAMAC Controller

Table 2.2: Power Consumption of the CPU board

Condition	Power consumption
booting time	16.8 W - 14.4 W
idle time	9.6 W - 7.2W
CAMAC access time	14.4 W - 12.0 W
Disk access time	16.8 W - 14.4 W

Table 2.3: Power Consumption of 6 V on CAMAC bus

Condition	Power consumption
idle time	1.8 - 2.4 W
CAMAC access time (data pattern = 0)	6.0 - 6.6 W
CAMAC access time (data pattern = 0xFFFFFF)	6.6 - 7.2 W

2.6 Power Consumption

The power consumption of the pipeline CAMAC controller depends on the running condition. The CPU board (C board) consumes lots of power. Its power is supplied via DC-DC converter from 24 V to 6 V on the D board. Thus, 24 V of CAMAC power supply should be monitored. On the other hand, CAMAC line driver consumes the power of 6 V on CAMAC bus. Thus, 6 V of CAMAC power supply should be also monitored. It is assumed that there is no graphic display. There were two CAMAC modules. One is the pipeline CAMAC controller. Another one is a switch register.

2.6.1 Power for CPU board

At booting time of Linux system, the current of 0.7-0.6 A flowed. This means that the power of 14.4 W to 16.8 W was consumed. Table 2.2 shows the power consumption depended on the condition. The power consumption in the table 2.2 except CAMAC access time only includes that of CPU board, not the switch register module because the module does not use 24V.

2.6.2 Power consumption of 6 V on CAMAC bus

There were two CAMAC modules. One is the pipeline CAMAC controller. Another one is a switch register. The power consumption in the table 2.3 is total power consumption including both modules.

Chapter 3

Operation

3.1 PCI I/O registers

There are three kinds of registers. One is for Tx and another is for Rx. The other is system register. Those registers are located in PCI I/O space and the size of those registers is 32-bit.

In programmed I/O (PIO), Data1 and Data2 registers for Tx and Rx are used for sending command frames and receiving reply frames, respectively. FIFO count register contains number of data in longword(4bytes). For an example, Rx FIFO count register is used for getting the number of data to be transferred.

In block I/O (DMA), Address register is used for pointing to address of kernel command/reply frame buffers and preset count register represents the number of frames in quadword(8 bytes) to be transferred. After the transfer, actual count register contains the number of frames in quadword(8 bytes) transferred actually. Control register and status register contains control and status information related to interrupt and DMA.

System register is used by system administrator.

Those registers are summarized in Table 3.1.

3.1.1 Tx Data1 and Tx Data2 registers

Data1 and Data2 registers for Tx are used for sending command frames in Programmed I/O mode. As the size of command frame is 64-bit, lower 32-bit corresponds to Data1 register and upper 32-bit corresponds to Data2 register. Table 3.2 shows the format. When the upper data is written into Data2 register, 64-bit command frame will be sent to CAMAC. If data in Data1 register is used again, next command frame will be sent by only writing data to Data2 register.

3.1.2 Tx Control register

Tx Control register contains control information related to interrupt and DMA. Whenever Tx FIFO is full, full-FIFO interrupt will occur if TC_INT_ENABLE_FULL_FIFO bit is set. Packet-end interrupt will occur only when Tx actual count register reaches to Tx present count register if TC_INT_ENABLE_PKT_END bit is set. When TC_CLR_FIFO bit is set, Tx FIFO in PCI control unit will be zero. Each bit assignment is shown in Table 3.3. The register can be read and written.

Table 3.1: PCI I/O register map

I/O offset address	Register name	Description
00h	TxData1	Tx data1 register
04h	TxData2	Tx data2 register
08h	TxControl	Tx control register
0Ch	TxStatus	Tx status register
10h	TxAddress	Tx address register
14h	TxPresetCount	Tx preset count register
18h	TxActualCount	Tx actual count register
1Ch	TxFifoCount	Tx FIFO count register
20h	RxData1	Rx data1 register
24h	RxData2	Rx data2 register
28h	RxControl	Rx control register
2Ch	RxStatus	Rx status register
30h	RxAddress	Rx memory address register
34h	RxPresetCount	Rx preset count register
38h	RxActualCount	Rx actual count register
3Ch	RxFifoCount	Rx FIFO count register
40h	System	System register
60h	IntData1	Int data1 register
64h	IntData2	Int data2 register
68h	IntControl	Int control register
6Ch	IntStatus	Int status register
7Ch	IntFifoCount	Int FIFO count register

Table 3.2: Tx Data1 and Tx Data2 registers

Bit assignment	Description
D31..D00	32-bit data used in Programmed I/O

Table 3.3: Tx Control register

Bit assignment	Bit name	Description
D31..D28		0(not used)
D27		0 (not used)
D26	TC_INT_ENABLE_FORCE_END	Enable Force-end interrupt
D25	TC_INT_ENABLE_FULL_FIFO	Enable full-FIFO interrupt
D24	TC_INT_ENABLE_PKT_END	Enable Packet-end interrupt
D23..D20		0(not used)
D19		0 (not used)
D18	TC_INT_CLR_FORCE_END	Clear Force-end interrupt
D17	TC_INT_CLR_FULL_FIFO	Clear full-FIFO interrupt
D16	TC_INT_CLR_PKT_END	Clear Packet-end interrupt
D15..D12		0(not used)
D11..D08		0(not used)
D07..D04		0(not used)
D03..D02		0 (not used)
D01	TC_CLR_FIFO	Clear Tx FIFO
D00	TC_SRT_DMA	Start DMA

3.1.3 Tx Status register

Tx Status register contains status information related to interrupt and DMA. When the Tx/Rx information is zero, the communication via CSP has no problem while non-zero of the information means communication error occurs. The Status register is read-only. The detail is summarized in Table 3.4.

3.1.4 Tx Memory Address register

In DMA, Tx Memory Address register is used for pointing to address of kernel command frame buffer to be transferred.

3.1.5 Tx Preset Count and Tx Actual Count registers

In DMA, Tx Preset Count register represents the number of command frames to be transferred. Tx Actual Count register contains the number of frames transferred actually. The maximum size of those registers are shown in Table 3.6, but the maximum number is limited by operating system. In Linux, maximum size of kernel buffer which is allocated in a continuous memory, is 128 KB. This means 16K frames can be sent at once. Therefore, the maximum size is 14-bit for Linux.

3.1.6 Tx FIFO Count register

Tx FIFO Count register represents the actual number of data in Tx FIFO. The unit is 32-bit, not 64-bit. The depth of FIFO is 256 for 128 command frames. Thus, the maximum size is 8-bit shown in Table 3.7.

Table 3.4: Tx Status register

Bit assignment	Bit name	Description
D31..D28	TS_RX_INFO	Rx information
D27..D24	TS_TX_INFO	Tx information
D23..D20		0(not used)
D19		0(not used)
D18	TS_INT_FORCE_END	When Force-end interrupt occurred, 1.
D17	TS_INT_FULL_FIFO	When full-FIFO interrupt occurred, 1.
D16	TS_INT_PKT_END	When Packet-end interrupt occurred, 1.
D15..D12		0(not used)
D11		0(not used)
D10	TS_FULL_FIFO	indicating Tx FIFO is full.
D09	TS_HALFFULL_FIFO	indicating Tx FIFO is half-full.
D08	TS_EMPTY_FIFO	indicating Tx FIFO is empty.
D07..D04		0(not used)
D03..D02		0(not used)
D01	TS_TIMEOUT_FRAME	No CAMAC response (in PIO)
D00	TS_DONE_FRAME	A frame was successfully processed.(in PIO)

Table 3.5: Tx Memory Address register

Bit assignment	Description
D31..D00	32-bit Memory address

Table 3.6: Tx Preset and Actual Count registers

Bit assignment	Description
D31..D00	24-bit Counter (actually 14-bit)

Table 3.7: Tx FIFO Count register

Bit assignment	Description
D31..D00	8-bit Counter

Table 3.8: Rx Data1 and Rx Data2 registers

Bit assignment	Description
D31..D00	32-bit data used in programmed I/o

Table 3.9: Rx Control register

Bit assignment	Bit name	Description
D31..D30		0(not used)
D29	RC_INT_ENABLE_FORCE_END	Enable Force-end interrupt
D28	RC_INT_ENABLE_PRESET_FIFO	Enable Preset interrupt
D27	RC_INT_ENABLE_FULL_FIFO	Enable full-FIFO interrupt
D26	RC_INT_ENABLE_HALFFULL_PKT	Enable half size Packet-end interrupt
D25	RC_INT_ENABLE_PKT_END	Enable Packet-end interrupt
D24	RC_INT_ENABLE_INPUT_FRAME	Enable first frame input interrupt
D23..D22		0(not used)
D21	RC_INT_CLR_FORCE_END	Clear Force-end interrupt
D20	RC_INT_CLR_PRESET_FIFO	Clear Preset interrupt
D19	RC_INT_CLR_FULL_FIFO	Clear full-FIFO interrupt
D18	RC_INT_CLR_HALFFULL_PKT	Clear half size of Packet-end interrupt
D17	RC_INT_CLR_PKT_END	Clear Packet-end interrupt
D16	RC_INT_CLR_INPUT_FRAME	Clear first frame input interrupt
D15..D012		0(not used)
D11..D08		0(not used)
D07..D04		0(not used)
D003..D02		0(not used)
D01	RC_CLR_FIFO	Clear Rx FIFO
D00	RC_SRT_DMA	Start DMA

3.1.7 Rx Data1 and Rx Data2 registers

Data1 and Data2 registers shown in Table 3.8 for Rx are used for receiving reply frames after sending command frames. When Data1 register is read, 64-bit reply frame will be stored into Data1 and Data2 registers. Data1 and Data2 registers contain lower 32-bit and upper 32-bit, respectively.

3.1.8 Rx Control register

Rx Control register contains control information related to interrupt and DMA. Whenever Rx FIFO is full, full-FIFO interrupt will occur if TC_INT_ENABLE_FULL_FIFO bit is set. Packet-end interrupt will occur only when Rx actual count register reaches to Rx present count register if TC_INT_ENABLE_PKT_END bit is set. When TC_CLR_FIFO bit is set, Rx FIFO in PCI control unit will be zero. Table 3.9 summaries them.

3.1.9 Rx Status register

Rx Status register contains status information related to interrupt and DMA. The Status register is read-only. The detail is summarized in Table 3.10.

Table 3.10: Rx Status register

Bit assignment	Bit name	Description
D31..D28	RS_RX_INFO	Rx information
D27..D24	RS_TX_INFO	Tx information
D23..D22		0(not used)
D21	RS_INT_FORCE_END	When Force-end interrupt occurred, 1.
D20	RS_INT_PRESET_FIFO	When Preset interrupt occurred, 1.
D19	RS_INT_FULL_FIFO	When full-FIFO interrupt occurred, 1.
D18	RS_INT_HALFFULL_PKT	When half size of Packet-end interrupt occurred, 1.
D17	RS_INT_PKT_END	When Packet-end interrupt occurred, 1.
D16	RS_INT_INPUT_FRAME	When first frame input interrupt occurred, 1.
D15..D12		0(not used)
D11		0(not used)
D10	RS_FULL_FIFO	indicating Rx FIFO is full.
D09	RS_HALFFULL_FIFO	indicating Rx FIFO is half-full.
D08	RS_EMPTY_FIFO	indicating Rx FIFO is empty
D07..D04		0(not used)
D03..D01		0(not used)
D00	RS_TIMEOUT_FRAME	Timeout in PIO

Table 3.11: Rx Memory Address register

Bit assignment	Description
D31..D00	32-bit Memory address

3.1.10 Rx Memory Address register

In DMA, Rx Memory Address register is used for pointing to address of kernel reply frame buffer to be transferred.

3.1.11 Rx Preset Count and Rx Actual Count registers

In DMA, Rx Preset Count register represents the number of reply frames to be transferred. Rx Actual Count register contains the number of frames transferred actually. The maximum size of those registers are shown in Table 3.12, but the maximum number is limited by operating system. In Linux, maximum size of kernel buffer which is allocated in a continuous memory, is 128 KB. This means 16K frames can be sent at once. Therefore, the maximum size is 14-bit for Linux.

Table 3.12: Rx Preset and Actual Count registers

Bit assignment	Description
D31..D00	24-bit Counter (actually 14-bit)

Table 3.13: Rx FIFO Count register

Bit assignment	Description
D31..D00	8-bit Counter

Table 3.14: System register

Bit assignment	Bit name	Description
D31	SYS_READY	CAMAC ready state
D30	SYS_RESET	reset PCI and CAMAC
D29..D28		0 (not used)
D27..D24	SYS_CAMAC_FRAME_SIZE	frame size in 2**N (read-only)
D23..D00		0 (not used)

3.1.12 Rx FIFO Count register

Rx FIFO Count register represents the actual number of data in Rx FIFO. The unit is 32-bit, not 64-bit. The depth of FIFO is 256 for 128 reply frames. the maximum size is 8-bit shown in Table 3.13.

3.1.13 System register

System register shown in Table 3.14 is used by system administrator. After power up, LED indicating system ready is blinking. When SYS_READY flag is set, the LED continues to be light. This means CAMAC controller can be used. If some trouble occurs and CAMAC controller hangs up, SYS_RESET flag is useful for resetting CAMAC controller. It is a reset without power off/on and resets the FPGAs of PCI and CAMAC. The frame size of command and reply is 64-bit, namely, 8 bytes. SYS_CAMAC_FRAME_SIZE represents the size as 2**N. Thus, the value should be 3. It is read-only.

3.1.14 Int Data1 and Int Data2 registers

Data1 and Data2 registers for Int are used for receiving interrupt reply frames in Programmed I/O mode. As the size of command frame is 64-bit, lower 32-bit corresponds to Data1 register and upper 32-bit corresponds to Data2 register. Table 3.15 shows the format. When Data1 register is read, 64-bit reply frame will be stored into Data1 and Data2 registers.

Table 3.15: Int Data1 and Int Data2 registers

Bit assignment	Description
D31..D00	32-bit data used in Programmed I/O

Table 3.16: Int Control register

Bit assignment	Bit name	Description
D31..D28		0(not used)
D27..D25		0 (not used)
D24	IC_INT_ENABLE	Enable interrupt for LAM & TRIG
D23..D20		0(not used)
D19..D17		0 (not used)
D16	IC_INT_CLR	Clear interrupt for LAM & TRIG
D15..D12		0(not used)
D11..D08		0(not used)
D07..D04		0(not used)
D03..D02		0 (not used)
D01	IC_CLR_FIFO	Clear Int FIFO
D00		0 (not used)

Table 3.17: Int Status register

Bit assignment	Bit name	Description
D31..D28		0(not used)
D27..D24		0(not used)
D23..D20		0(not used)
D19..D17		0(not used)
D16	IS_INT	When an interrupt occurred, 1.
D15..D12		0(not used)
D11		0(not used)
D10	IS_FULL_FIFO	indicating Int FIFO is full.
D09	IS_HALFFULL_FIFO	indicating Int FIFO is half-full.
D08	IS_EMPTY_FIFO	indicating Int FIFO is empty.
D07..D04		0(not used)
D03..D01		0(not used)
D00	IS_TIMEOUT_FRAME	Timeout in PIO

3.1.15 Int Control register

Int Control register contains control information related to LAM & TRIG interrupt. The interrupt will occur when an interrupt reply frame comes into and IC_INT_ENABLE bit is set. When IC_CLR_FIFO bit is set, Int FIFO will be zero. Each bit assignment is shown in Table 3.16. The register can be read and written.

3.1.16 Int Status register

Int Status register contains status information related to LAM & TRIG interrupt. The Status register is read-only. The detail is summarized in Table 3.17.

3.1.17 Int FIFO Count register

Int FIFO Count register represents the actual number of data in Int FIFO. The unit is 32-bit, not 64-bit. The depth of FIFO is 256 for 128 command frames. Thus, the maximum

Table 3.18: Int FIFO Count register

Bit assignment	Description
D31..D00	8-bit Counter

Table 3.19: Frame Format

D63..D56	D55..D00
Frame Header (8 bits)	Payload (56 bits)

size is 8-bit shown in Table 3.18.

3.2 Frame Format

There are two types of frame, namely, command frame and reply frame. The command frame is sent to CAMAC from memory in computer via PCI bus while the reply frame is generated at CAMAC and then stored into memory in computer via PCI bus. The frame shown in Table 3.19 consists of the frame header and the payload. The size of the header is a byte or 8 bits while that of the payload is 7 bytes or 56 bits. The 64-bit frame are divided into two 32-bit data. The lower 32-bit data (D31..D00) corresponds to Data1 register and first 32-bit data in 32-bit data array. The upper 32-bit data (D63..D32) corresponds to Data2 register and second 32-bit data in 32-bit data array. This means the data format is represented in so-called little endian.

3.2.1 Frame Header

A frame includes a frame header of a byte shown in Table 3.20. A packet can include at least one or more frames. The CC_SPKT bit should be set for the start frame in a packet. The CC_EPKT bit should be set for the end frame in a packet. When both the bits are set, the packet has a single frame. For the other frame, both the bits should be reset.

CC_SEL bits chooses the operation in Table 3.21.

Table 3.22 shows all types of frame header.

Table 3.20: Frame Header

Bit assignment	Bit name	Description
D63	CC_FIXED	always 1
D62	CC_SPKT	1 if start frame in a packet
D61	CC_EPKT	1 if end frame in a packet
D60..D59	CC_SEL	select operation
D58..D56		0 (not used)

Table 3.21: Selection of Operation

Bit 60	Bit 59	Operation
0	0	Basic CAMAC operation
0	1	CAMAC LAM Interrupt operation
1	0	Basic DAQ operation
1	1	DAQ Trigger Interrupt operation

Table 3.22: Frame Header Contents

Operation	Start Frame	Normal Frame	End Frame	Packet Frame
Basic CAMAC	0xC0	0x80	0xA0	0xE0
CAMAC LAM	-	-	-	0xE8
Basic DAQ	0xD0	0x90	0xB0	0xF0
DAQ TRIG	-	-	-	0xF8

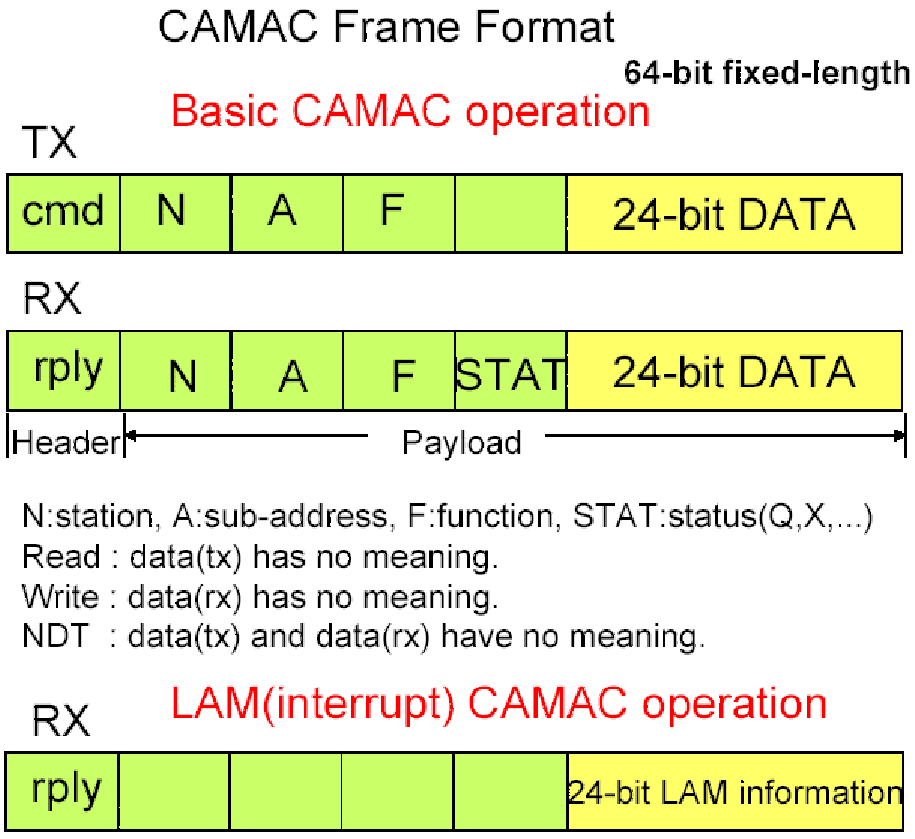

Figure 3.1: Frame formats of CAMAC command and reply

Table 3.23: Payload for Basic CAMAC command frame

Bit assignment	Bit name	Description
D55..D48	CCEXE_N	CAMAC station number
D47..D40	CCEXE_A	CAMAC sub-address
D39..D32	CCEXE_F	CAMAC function
D31..D24		0 (not used)
D23..D00	CCEXE_WDATA	Data to be written

Table 3.24: Payload for Basic CAMAC reply frame

Bit assignment	Bit name	Description
D55..D48	CCEXE_N	CAMAC station number
D47..D40	CCEXE_A	CAMAC sub-address
D39..D32	CCEXE_F	CAMAC function
D31..D24	CCEXE_STAT	CAMAC status
D23..D00	CCEXE_RDATA	Read Data

3.2.2 Basic CAMAC function

Figure 3.1 shows frame formats of CAMAC command and reply. It also includes CAMAC Interrupt reply frame. For the basic CAMAC function, the command frame consists of **cmd** as the frame header, station number **N**, sub-address **A**, function **F** and 24-bit data **24-bit DATA** while the reply frame consists of **reply** as the frame header, station number **N**, sub-address **A**, function **F**, status **STAT** and 24-bit data **24-bit DATA**. Table 3.23 and 3.24 show the contents of payloads for the command frame and the reply frame, respectively.

Table 3.25 shows the contents of CAMAC status in the reply frame.

3.2.3 CAMAC LAM Interrupt

Figure 3.1 shows the frame format of the CAMAC LAM interrupt reply frame. It consists of **reply** as the frame header and **24-bit LAM pattern**. If the bit 0 of the LAM pattern is set, it means that the CAMAC module at the CAMAC station 1 generates a LAM

Table 3.25: CAMAC status

Bit assignment	Bit name	Description
D31	CCEXE_STAT_FSTC	Fast Cycle mode
D30		0 (not used)
D29	CCEXE_STAT_LSUM	indicating LAM exists
D28	CCEXE_STAT_IE	indicating LAM interrupt is enable
D27		0 (not used)
D26	CCEXE_STAT_INHIBIT	indicating CAMAC INHIBIT is set
D25	CCEXE_STAT_X	CAMAC X
D24	CCEXE_STAT_Q	CAMAC Q

Table 3.26: Payload for CAMAC LAM Interrupt reply frame

Bit assignment	Bit name	Description
D55..D24		0 (not used)
D23..D00	CCLAM_PATTERN	Read LAM pattern

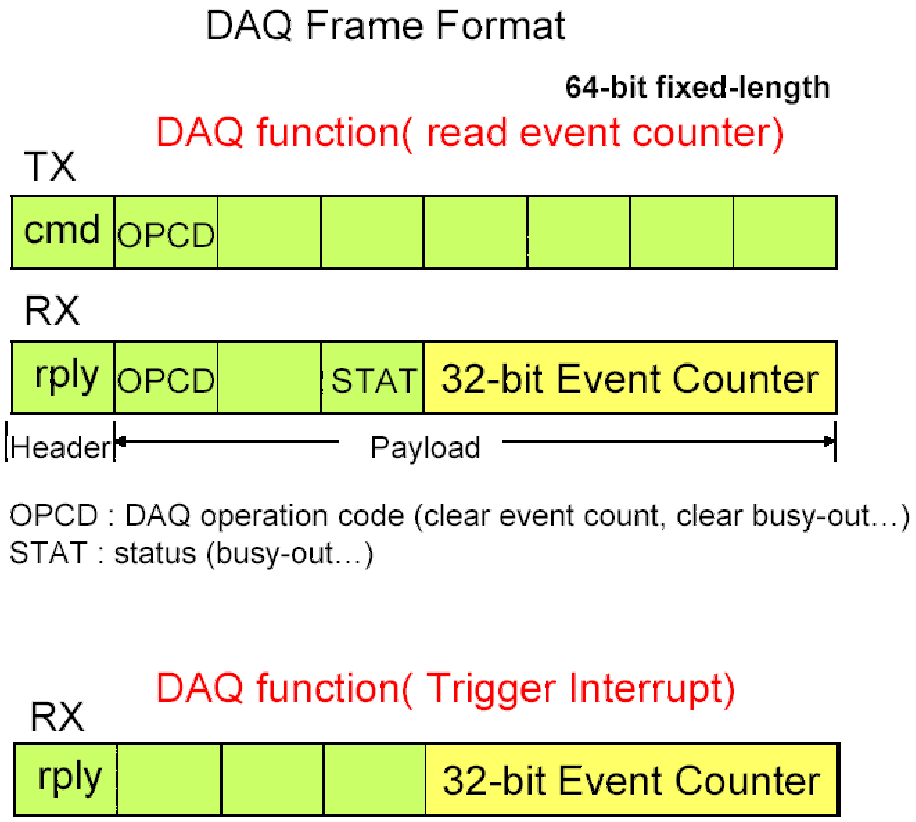

Figure 3.2: Frame formats of DAQ command and reply

interrupt while that at the station 23 generates the interrupt if the bit 22 is set.

3.2.4 DAQ function

Figure 3.2 shows frame formats of DAQ command and reply. It also includes DAQ Trigger Interrupt reply frame. For the DAQ function, the command frame consists of **cmd** as the frame header and a control byte **OPCD**. The reply frame consists of **reply** as the frame header, a DAQ control byte **OPCD**, a DAQ status byte **STAT** and **32-bit Event Counter**. Table 3.27 and 3.29 show the contents of payloads for the command frame and the reply frame, respectively.

Table 3.28 shows the contents of a DAQ control byte. When the value 2 is set, the event counter becomes zero. After enabling the DAQ trigger interrupt by setting 3 into the value, the event counter will increment if the interrupt is input from TRIG-IN LEMO connector at the front panel of the pipeline CAMAC controller. At the moment,

Table 3.27: Payload for Basic DAQ command frame

Bit assignment	Bit name	Description
D55..D48	DAQEXE_CTRL	DAQ Control bits
D47..D40		0 (not used)
D39..D00		0 (not used)

Table 3.28: DAQ control

Value of OPCD	Value name	Description
0x00	DAQEXE_CTRL_READ	Read Event Counter
0x01	DAQEXE_CTRL_CLRBSY	Clear Busy-Out signal
0x02	DAQEXE_CTRL_CLRCNT	Clear Event Counter
0x03	DAQEXE_CTRL_ENABLE	Enable DAQ trigger interrupt
0x04	DAQEXE_CTRL_DISABLE	Disable DAQ trigger interrupt (default)
0x05	DAQEXE_CTRL_PLSOUT	Enable output pulse at BSY-OUT connector
0x06	DAQEXE_CTRL_NOFRAME	Disable generating interrupt frame (default)
0x0A	DAQEXE_CTRL_WAITTRIG	Wait for a trigger interrupt

the Busy-Out signal will be also generated at BUSY-OUT LEMO connector at the front panel. After that, the DAQ trigger interrupt does not occur even if the interrupt is input. To enable next interrupt, Busy-Out should be reset by setting 1 into the value.

If DAQEXE_CTRL_PLSOUT is set, the behavior of the signal generation at at BSY-OUT connector will change. When a DAQ trigger occurs, Busy-Out signal is not generated. Instead, a output pulse will be generated when DAQEXE_CTRL_CLRBSY bit is set. The pulse width is 40 nsec.

The aim of this mode is as follows; If the normal mode is used, BUSY-OUT signal will be generated and used for veto to next trigger. However, the timing of the trigger input and the BUSY-OUT slightly change. For arranging the timing precisely, another veto signal will be used instead of the BUSY-OUT signal in normal mode. For setting the veto off, this mode is used. When clearing BUSY-OUT, a 40 nsec pulse will be generated. It sets the veto off.

A DAQ status **DAQEXE_STAT_TRIGIN** indicates the existence of a DAQ trigger input. The bit will be set when the trigger is input even if the trigger is not enabled.

Table 3.29: Payload for Basic DAQ reply frame

Bit assignment	Bit name	Description
D55..D48	DAQEXE_CTRL	DAQ Control bits
D47..D40		0 (not used)
D39..D32	DAQEXE_STAT	DAQ Status bits
D31..D00	DAQEXE_COUNT	Read Event Counter

Table 3.30: DAQ Status

Bit assignment	Bit name	Description
D39..D36		0 (not used)
D35	DAQEXE_STAT_PLSOUT	Pulse-out mode
D34	DAQEXE_STAT_TRIGIN	DAQ trigger input
D33	DAQEXE_STAT_ENTRIGIN	Enable Trigger interrupt
D32	DAQEXE_STAT_BSY	Busy-Out

Table 3.31: Payload for DAQ Trigger Interrupt reply frame

Bit assignment	Bit name	Description
D55..D32		0 (not used)
D31..D00	DAQINT_COUNT	Event Counter

3.2.5 DAQ Trigger Interrupt

Figure 3.2 shows frame formats of DAQ trigger interrupt reply frame. Table 3.31 shows the contents of the payload. The reply frame consists of **reply** as the frame header and the **32-bit event counter**.

3.3 Operation procedure

In the following sub-section, basic operation procedures of PIO, DMA and Interrupt to CPU are described. They do not explain real programs used in the device driver, but just show how to manipulate the PCI registers briefly.

3.3.1 PIO

For PIO, Data1 and Data2 registers for Tx and Rx are used for the data path. The Status and FifoCount registers for Tx and Rx are used for the control path. It is assumed that TxFifoCount and RxFifoCount are zero. This means there is no frame in the TxFIFO and the RxFIFO. The lower 32-bit word of command frame should be put into TxData1 register first. At the moment, the command frame is not sent to the CAMAC control unit. When upper 32-bit word of the command frame is put into TxData2 register, the command frame will be sent to CAMAC via CSP. If the transfer to CAMAC is done successfully, TS_DONE_FRAME bit in TxStatus register will be set. Otherwise, TS_TIMEOUT_FRAME will be set. This means there is no CAMAC response. After the CAMAC execution, the reply frame will be stored into the RxFIFO in the PCI control unit. At the moment, the contents of RxFifoCount will be a value 2. This means there is a reply frame in the RxFIFO. For the read-out, RxData1 register should be read first. It includes the lower 32-bit word of reply frame. At the same time, the upper 32-bit word of the reply frame will be latched into the RxData2 register. When the data are read out, RxFifoCount will be decremented by 2. Thus, the upper 32-bit word can be read from the RxData2 register anytime. When the RxFIFO is empty and the RxData1 register is

read, RS_TIMEOUT_FRAME bit in the RxStatus register will be set. This means the operation failed.

3.3.2 DMA

There are several registers related to DMA operation to pipeline CAMAC controller. The physical addresses of data buffers in Linux kernel are set into Address registers for Tx and Rx. The frame counts are set into PresetCount registers for Tx and Rx. Control registers for Tx and Rx are used to start the DMA operation, by setting TC_SRT_DMA bit in TxControl register and RC_SRT_DMA bit in RxControl register. It is assumed that TxFifoCount and RxFifoCount are zero. The following procedure is efficient for the DMA operation. The DMA operation for Rx should start first. At the moment, nothing happens because there is no reply frame in the RxFIFO. After starting the DMA, do not wait for the completion of the DMA. Instead, Tx DMA operation should start next. After the Tx DMA operation starts, command frames will be sent to CAMAC control unit. the control unit executes the command frames and then sends the reply frames to PCI control unit. As the result, the reply frames will be sent to Linux kernel buffer as soon as they are stored into the RxFIFO, because Rx DMA operation already starts.

3.3.3 Interrupt

There are two groups of interrupt sources. One is for DMA operations. Another is for external interrupt sources. Enabling, disabling and clearing the interrupts are done on Control registers for Tx, Rx and Int while the Status registers shows current status of the interrupt sources.

There are lots of interrupt sources for the DMA operation. An important interrupt is packet end interrupt. The interrupt occurs when the PresetCount register exhausts or the value becomes zero. The related bits in TxControl register are TC_INT_ENABLE_PKT_END bit in TxControl register for enabling the packet end interrupt of Tx and TC_CLR_PKT_END bit in the register for clearing it. To disable the interrupt, TC_INT_ENABLE_PKT_END bit should be reset. The related bits in Rx Control register are similar to that in Tx Control register.

For external interrupt sources, the interrupt occurs when a reply frame comes into the IntFIFO first after the contents of the IntFifoCount register is zero. When IC_INT_ENABLE bit in IntControl register is set and an interrupt reply frame comes into the IntFIFO, the interrupt occurs.

3.4 Special CAMAC Functions to pipeline CAMAC controller

There are several CAMAC functions to CAMAC controller in Table 3.32. At Power-on reset, those conditions are following. CAMAC Inhibit is set. Interrupt to CPU is disabled. Fast Cycle is disabled. All bits in LAM Enable register are reset.

Fast Cycle mode ignores CAMAC S2 timing. When the mode is set, the CAMAC cycle becomes 0.72 usec instead of 1.04 usec. Many CAMAC modules can work without

Table 3.32: Special CAMAC Function

Function	**Description**
N(25).A(0).F(24)	Clear CAMAC Inhibit
N(25).A(0).F(26)	Set CAMAC Inhibit
N(25).A(1).F(24)	Disable Interrupt to CPU
N(25).A(1).F(26)	Enable Interrupt to CPU
N(25).A(2).F(24)	Disable Fast Cycle
N(25).A(2).F(26)	Enable Fast Cycle
N(25).A(0).F(16)	Generate CAMAC C
N(25).A(0).F(17)	Generate CAMAC Z
N(25).A(1).F(0)	Read LAM Enable register
N(25).A(1).F(16)	Write LAM Enable register

CAMAC S2 timing, but it is invalid operation for CAMAC protocol. The mode should be used carefully.

Chapter 4

CAMAC Device Driver and the Library

The CAMAC device driver is developed for only the pipeline CAMAC controller. There are two kinds of CAMAC library developed for the controller. One is a general purpose CAMAC library and a CAMAC library dedicated to the pipeline CAMAC controller.

4.1 Installation

The CAMAC device driver and the library are pre-installed at the purchase of the CAMAC controller. The following instruction is not necessary if the Linux system is not destroyed. The instruction is provided for the recovery process.

4.1.1 How to get the distribution kit

The device driver and the library is available at the following URLs.

- KEK online group : http://www-online.kek.jp/~ yasu/Parallel-CAMAC/
- TOYO corporation : http://www.toyo.co.jp/daq/index.html

4.1.2 How to compile and load the device driver

- Make a device driver and test programs.
  The major number of the device is assumed to be 70. Thus, If it is not convenient, **Makefile** file should be modified.

  ```
  % cd /home/toyo/camac
  % make clean; make
  ```

- Make the device files.
  This makes a device file in /dev directory. The name is pcc0.

  ```
  % su
  # make device
  ```

- install the device driver in Linux kernel.

```
# make install
```

- load the device driver.

```
# /sbin/insmod pcc
```

- Make sure if it is successfully loaded.
  If you can find, a message "PCC has been installed.", it's OK.

```
% /bin/dmesg
```

4.2 General Purpose CAMAC Library

The library is a general purpose CAMAC library. The user interface is common for all CAMAC libraries KEK online group supports [17], [18], [19], [20], [21]. The CAMAC library consists of the following function groups.

- Setup Functions
- CAMAC Functions
- CAMAC Single Actions
- Interrupt Handling

4.2.1 Setup Functions

camopen

- **SYNOPSIS**
  #include "camlib.h"

  int camopen(int crate);
  int CAMOPEN(int crate);
- **DESCRIPTION**
  camopen opens a CAMAC port to enable access.
- **RETURN VALUE**
  On success of **camopen**, zero is returned. If failed, a negative value is returned.

camcls

- **SYNOPSIS**
  #include "camlib.h"

  int camcls(int crate);
  int CAMCLS(int crate);

- **DESCRIPTION**
  camcls closes the CAMAC port to disable access. **crate** is the number of crate to be opened. For the pipeline CAMAC controller, it is not used.

- **RETURN VALUE**
  On success of **camcls**, zero is returned. If failed, a negative value is returned.

4.2.2 CAMAC Functions

cgenz

- **SYNOPSIS**
  #include "camlib.h"

  int cgenz(int crate);
  int CGENZ(int crate);

- **DESCRIPTION**
  cgenz initializes the CAMAC port. **crate** specifies the crate to be accessed. For the pipeline CAMAC controller, it is not used.

- **RETURN VALUE**
  On success of **cgenz**, zero is returned. If failed, a negative value is returned.

cgenc

- **SYNOPSIS**
  #include "camlib.h"

  int cgenc(int crate);
  int CGENC(int crate);

- **DESCRIPTION**
  cgenc clears the CAMAC port. **crate** specifies the crate to be accessed. For the pipeline CAMAC controller, it is not used.

- **RETURN VALUE**
  On success of **cgenc**, zero is returned. If failed, a negative value is returned.

cseti

- **SYNOPSIS**
  #include "camlib.h"

  int cseti(int crate);
  int CSETI(int crate);

- **DESCRIPTION**
  cseti sets CAMAC INHIBIT on the CAMAC port. **crate** specifies the crate to be accessed. For the pipeline CAMAC controller, it is not used.

- **RETURN VALUE**
  On success of **cseti**, zero is returned. If failed, a negative value is returned.

cremi

- **SYNOPSIS**
  #include "camlib.h"

  int cremi(int crate);
  int CREMI(int crate);

- **DESCRIPTION**
  cremi removes CAMAC INHIBIT on the CAMAC crate. **crate** specifies the crate to be accessed. For the pipeline CAMAC controller, it is not used.

- **RETURN VALUE**
  On success of **cremi**, zero is returned. If failed, a negative value is returned.

4.2.3 CAMAC Single Action

camac

- **SYNOPSIS**
  #include "camlib.h"

  int camac(int crate, int n, int a, int f, int *data, int *q, int *x);
  int CAMAC(int crate, int n, int a, int f, int *data, int *q, int *x);

- **DESCRIPTION**
  camac accesses CAMAC modules in the CAMAC crate. **crate** specifies the number of crate to be accessed. For the pipeline CAMAC controller, it is not used. There are CAMAC station number **n**, CAMAC sub-address **a**, CAMAC function **f** and data **data**. After calling the **camac**, the values of CAMAC Q **q** and CAMAC X **x** are returned.

- **RETURN VALUE**
  On success of **camac**, zero is returned. If failed, a negative value is returned.

4.2.4 Interrupt Handling

cenlam

- **SYNOPSIS**
  #include "camlib.h"

  int cenlam(int crate, int mask);
  int CENLAM(int crate, int mask);

- **DESCRIPTION**
  cenlam enables interrupt to CPU. **crate** specifies the crate to be accessed. For the pipeline CAMAC controller, it is not used. When the bit of the value **mask** is 1, it enables the corresponding CAMAC modules interrupt to CPU. The bit 0 corresponds to the CAMAC station 1 while the bit 23 does the station 24.

- **RETURN VALUE**
  On success of **cenlam**, zero is returned. If failed, a negative value is returned.

cdslam

- **SYNOPSIS**
  #include "camlib.h"

  int cdslam(int crate):
  int CDSLAM(int crate):

- **DESCRIPTION**
  cdslam disables interrupt to CPU. **crate** specifies the crate to be accessed. For the pipeline CAMAC controller, it is not used.

- **RETURN VALUE**
  On success of **cdslam**, zero is returned. If failed, a negative value is returned.

cwtlam

- **SYNOPSIS**
  #include "camlib.h"

  int cwtlam(int crate, int timeout):
  int CWTLAM(int crate, int timeout):

- **DESCRIPTION**
  cwtlam waits for an interrupt from a CAMAC module. **crate** specifies the crate to be accessed. For the pipeline CAMAC controller, it is not used. The time-out time **timeout** is specified in unit of clock tick (normally, it 10 msec, but it depends on Linux kernel configuration). When the value is less than 1, the default value is applied.

- **RETURN VALUE**
  On success of **cwtlam**, zero is returned. If failed, a negative value is returned.

4.3 CAMAC Library dedicated to pipeline CAMAC controller

There are five kinds of subroutine groups in the CAMAC library.

- CAMAC/DAQ command frame generators
- PIO executor
- Block I/O executor
- Wait routines for Trigger and LAM Interrupts
- CAMAC/DAQ reply frame Extractor

CAMAC/DAQ command frame generators generates CAMAC or DAQ command frame by giving CAMAC or DAQ parameters such as N (station number), A (sub-address), F (function) and so on. Those command frames are stored into a user-specified buffer with a CAMAC frame buffer structure.

PIO executor executes command frames in the buffer in Programmed I/O. The executor gets command frames from the buffer and then stores reply frames into another user-specified buffer.

Block I/O executor executes the frames in the buffer in DMA mode. The executor gets command frames from the buffer and then stores reply frames into another user-specified buffer.

"Wait routines for Trigger and LAM Interrupts" is for waiting an interrupt from CAMAC or DAQ.

CAMAC/DAQ reply frame Extractor extracts data and CAMAC status such as Q and X from the buffer including reply frames.

4.3.1 CAMAC frame buffer structure

The CAMAC frame buffer structure is shown in Figure 4.1.

Table 4.1: CAMAC frame buffer structure

Offset(32-bit)	Item	Description
0	Total buffer length	Unit is 64-bit.
1	Actual buffer length of frames	Unit is 64-bit word
2	Start address of frames	
—	—	
N	End address of the frame buffer	

The unit of data in the frame buffer is 32-bit (4 bytes) while the size of frame is 64-bit (8 bytes). This means the actual buffer length increases 2 when a frame is stored into the buffer. There are four types of frame, a first CAMAC/DAQ frame, a normal CAMAC/DAQ frame, a end CAMAC/DAQ frame and a packet CAMAC/DAQ frame. The packet CAMAC frame includes a CAMAC/DAQ frame in a packet. The packet

can include multiple CAMAC/DAQ frames while the first frame should be the first CAMAC/DAQ frame, the end frame should be the end CAMAC/DAQ frame and the normal frame should be between the first frame and the end frame.

4.3.2 CAMAC/DAQ command frame generators

There are two kinds of subroutines for generating frames for CAMAC and DAQ.

cam_gen_init

- **SYNOPSIS**
  #include "pcc.h"

  int cam_gen_init(int length, int* buf);

- **DESCRIPTION**
  cam_gen_init() initializes the frame buffer **buf**. Actually, the total buffer length will be filled with **length** the buffer and the actual buffer length will be filled with zero. **length** is specified with unit of 64-bit frame size.

- **RETURN VALUE**
  cam_gen_init() always returns zero as success.

- **ERRORS** There is no error.

cam_gen_cc

- **SYNOPSIS**
  #include "pcc.h"

  int cam_gen_cc(int *buf, int n, int a, int f, int data);

- **DESCRIPTION**
  cam_gen_cc() generates a CAMAC frame from **n**, **a**, **f** and **data** and then stores the frame into **buf**. The start/end bits in the frame are automatically generated.

- **RETURN VALUE**
  On success of **cam_gen_cc()**, zero is returned. If **n**, **a** or **f** is invalid, -1 is returned. When there is no space to store the frame in the frame buffer, -2 is returned.

cam_gen_daq

- **SYNOPSIS**
  #include "pcc.h"

  int cam_gen_daq(int* buf, int cmd);

- **DESCRIPTION**
  cam_gen_daq() generates a DAQ frame from **cmd**. The start/end bits in the frame are automatically generated.

- **RETURN VALUE**
  On success of **cam_gen_daq()**, zero is returned. If **cmd** is invalid, -1 is returned. When there is no space to store the frame in the frame buffer, -2 is returned.

4.3.3 CAMAC open/close

cam_open

- **SYNOPSIS**
  #include "pcc.h"

  int cam_open(void);

- **DESCRIPTION**
  cam_open() opens a CAMAC device and gets the file descriptor.

- **RETURN VALUE**
  On success of **cam_open()**, the file descriptor is returned. If -1 is returned, it is open error.

cam_close

- **SYNOPSIS**
  #include "pcc.h"

  int cam_close(int fd);

- **DESCRIPTION**
  cam_close() closes the CAMAC device.

- **RETURN VALUE**

4.3.4 PIO routines

PIO stands for Programmed Input Output. Those routines does not use Direct Access Method (DMA). Instead, CPU puts command frames and then gets reply frames. This method is simple and less of the initiation overhead while it consumes CPU power and its speed is rather than that of DMA.

cam_put

- **SYNOPSIS**
  #include "pcc.h"

  int cam_put(int fd, int data, int cmd);

- **DESCRIPTION**
  cam_put() writes a command frame from **data** and **cmd** into the CAMAC controller. The operation checks Tx FIFO count. If the count is full and timeout count exhausts, it fails.

- **RETURN VALUE**
  On success of **cam_put**, zero is returned. Otherwise, -1 is returned.

cam_get

- **SYNOPSIS**
  #include "pcc.h"

  int cam_get(int fd, int* data, int* rply);

- **DESCRIPTION**
  cam_get() reads a reply frame from the CAMAC controller and then stores into **data** and **reply**. The operation checks if Rx FIFO count is empty or not. If the count is empty and timeout count exhausts, it fails.

- **RETURN VALUE**
  On success of **cam_read_pio**, zero is returned. Otherwise, -1 is returned.

cam_exec_pio

- **SYNOPSIS**
  #include "pcc.h"

  int cam_exec_pio(int fd, int* cmdbuf, int* rplybuf);

- **DESCRIPTION**
  cam_exec_pio() executes command frames in the frame buffer **cmdbuf** and then stores the reply frames into the frame buffer **rplybuf**. The operation checks Rx FIFO and Tx FIFO. The operation on Tx and Rx are done concurrently. When Tx FIFO is not full and the command frames to be transferred remains, the command frame is sent while the reply frames is read when Rx FIFO is not empty. The timeout is checked. The maximum number of command frames to be transferred is 16K (1K=1024) frames.

- **RETURN VALUE**

cam_single_cc

- **SYNOPSIS**
  #include "pcc.h"

  int cam_single_cc(int fd, int n, int a, int f, int *data, int *q, int *x);

- **DESCRIPTION**
  cam_single_cc() executes a single CAMAC frame specified by station number **n**, sub-address **a**, function **f** and data **data**. It returns data, **q** and **x**.

- **RETURN VALUE**

cam_single_daq

- **SYNOPSIS**
  #include "pcc.h"

  int cam_single_daq(int fd, int func, int *data);

- **DESCRIPTION**
  cam_single_daq() executes a single DAQ frame specified by DAQ function **func**. It returns **data** if required.

- **RETURN VALUE**

4.3.5 Block I/O routines

Block I/O routes use DMA. After initiating PCI master engine on PCI control unit of CAMAC controller, the engine transfers data between the controller and CPU memory, without CPU intervention. The advantage of DMA is that it gains high throughput, but the initiation overhead is rather larger than that of PIO.

cam_exec_dma

- **SYNOPSIS**
  #include "pcc.h"

  int cam_exec_dma(int fd, int* cmdbuf, int* rplybuf);

- **DESCRIPTION**
  cam_exec_dma executes the command frames in the command frame buffer **cmdbuf** and then stores the results into the reply frame buffer **rplybuf**. **cam_exec_dma** issues read function first and then write function. As the result, maximum command frames(16k frames) can be sent and maximum reply frames (16k frames) can be received at once.

- **RETURN VALUE**

cam_exec_dma_seq

- **SYNOPSIS**
  #include "pcc.h"

int cam_exec_dma_seq(int fd, int* cmdbuf, int* rplybuf);

- **DESCRIPTION**
  cam_exec_dma_seq executes the command frames in the command frame buffer **cmdbuf** and then stores the results into the reply frame buffer **rplybuf**. **cam_exec_dma_seq** issues write function and read function sequentially. As the result, maximum FIFO frames (120 frames) can be sent and maximum reply frames (120 frames) can be received at once. Therefore, **cam_exec_dma_seq** repeats the above sequence up to the specified number of command frames.

- **RETURN VALUE**

4.3.6 Combined routine

Combined routine uses not only DMA but also PIO. The executor takes both advantages of DMA and PIO. In small number of frames to be executed, the executor chooses PIO method while it chooses DMA in the large number of frames. The cross point is determined by the performance measurement.

cam_exec

- **SYNOPSIS**
  #include "pcc.h"

  int cam_exec(int fd, int* cmdbuf, int* rplybuf);

- **DESCRIPTION**
  cam_exec() executes command frames in the frame buffer **cmdbuf** and then stores the reply frames into the frame buffer **rplybuf**. **cam_exec()** switches the transfer mode according to the specified number of command frames. If the number reaches NUM_FRAME_SWITCH, **cam_exec()** will switch the transfer from in PIO mode to in DMA mode.

- **RETURN VALUE**

4.3.7 Interrupt handling routines for Trigger and LAM Interrupts

- **SYNOPSIS**
  #include "pcc.h"

  int cam_enable_lam(int fd, int enable_pattern);

- **DESCRIPTION**
  cam_enable_lam() enables CAMAC LAM interrupt. **enable_pattern** is used for individual stations. The bit 0 corresponds to CAMAC station 1 while the

bit 22 corresponds to the station 23. This routine should be called just before **cam_wait_lam()**.

- **RETURN VALUE**

- **SYNOPSIS**
  #include "pcc.h"

  int cam_disable_lam(int fd);

- **DESCRIPTION**
  cam_disable_lam() disables CAMAC LAM interrupt.

- **RETURN VALUE**

- **SYNOPSIS**
  #include "pcc.h"

  int cam_wait_lam(int fd, int* lam_pattern, int timeout);

- **DESCRIPTION**
  cam_wait_lam() waits for a CAMAC LAM interrupt. If the interrupt occurs, the LAM pattern will be stored into **lam_pattern**.

- **RETURN VALUE**

- **SYNOPSIS**
  #include "pcc.h"

  int cam_enable_trig(int fd);

- **DESCRIPTION**
  cam_enable_trig() enables event trigger. This routine should be called just before **cam_wait_trig()**.

- **RETURN VALUE**

- **SYNOPSIS**
  #include "pcc.h"

  int cam_disable_trig(int fd);

- **DESCRIPTION**
  cam_disable_trig() disables event trigger.

- **RETURN VALUE**

- **SYNOPSIS**
  #include "pcc.h"

  int cam_wait_trig(int fd, int* event_count);

- **DESCRIPTION**
  cam_wait_trig() waits for an event trigger. If the interrupt occurs, current event counter will be stored into **event_count**.

- **RETURN VALUE**

4.3.8 CAMAC/DAQ reply frame extraction routines

cam_extract_cc_data

- **SYNOPSIS**
  #include "pcc.h"

  int cam_extract_cc_data(int* rplybuf, int len, int* actuallen, int* data);

- **DESCRIPTION**
  cam_extract_cc_data() extracts array of data from a reply frame buffer **rplybuf** and then stores the data into the array **data**. The specified **len** is a total length of data array **data**. Not only data but also the actual length **actuallen** of frames in the frame buffer are returned.

- **RETURN VALUE**

cam_extract_cc_status

- **SYNOPSIS**
  #include "pcc.h"

  int cam_extract_cc_status(int* rplybuf, int len, int* actuallen, int* status);

- **DESCRIPTION**
  cam_extract_cc_status() extracts array of status including CAMAC Q and CAMAC X from a reply frame buffer **rplybuf**. The specified **len** is a total length of status array **status**. Not only status but also the actual length **actuallen** of frames in the frame buffer are returned.

- **RETURN VALUE**

cam_extract_cc_qx

- **SYNOPSIS**
  #include "pcc.h"

int cam_extract_cc_qx(int status, int* q, int* x);

- **DESCRIPTION**
  cam_extract_cc_qx extracts CAMAC Q **q** and CAMAC X **x** from a status **status**.
- **RETURN VALUE**

cam_extract_daq_data

- **SYNOPSIS**
  #include "pcc.h"

  int cam_extract_daq_data(int* rplybuf, int len, int* actuallen, int* data);
- **DESCRIPTION**
  cam_extract_daq_data() extracts array of data from a reply frame buffer **rplybuf** and then stores the data into the array **data**. The specified **len** is a total length of data array **data**. Not only data but also the actual length **actuallen** of frames in the frame buffer are returned.
- **RETURN VALUE**

4.4 Examples

4.4.1 Tools

rst_cam command

The **rst_cam** command resets FPGAs in the pipeline CAMAC controller. The command does not clear the FIFOs.

```
1   % ./rst_cam
```

clr_fifo command

The **clr_fifo** clears all FIFOs in the pipeline CAMAC controller.

```
1   % ./clr_fifo
```

dump_reg command

The **dump_reg** command dumps contents of PCI I/O registers. It does not include contents of Tx Data1, Tx Data2, Rx Data1 and Rx Data2.

```
1   % ./dump_reg
2   Tx Control      = 0
3   Tx Status       = 1
4   Tx Address      = f680000
5   Tx Preset Count = a
6   Tx Actual Count = 0
7   Tx Fifo Count   = 0
8   Rx Control      = 0
9   Rx Status       = 1
10  Rx Address      = f640000
11  Rx Preset Count = a
12  Rx Actual Count = a
13  Rx Fifo Count   = 0
14  System          = 83000000
```

cam command

The **cam** command is a simple CAMAC access command using PIO. The procedure is as follows;

```
1   % ./cam
2   usage : ./cam n a f [data]
3   % ./cam 4 0 16 0xffffff
4   Q = 1 : X = 1
5   % ./cam 4 0 0
6   Q = 1 : X = 1 : data = ffffff
7   % ./cam 4 0 16 0xaaaaaa
8   Q = 1 : X = 1
9   % ./cam 4 0 9
10  Q = 0 : X = 1
```

The **n** is CAMAC station number. **a** is CAMAC sub-address. **f** is CAMAC function. **data** has 24-bit width and it should be put in Hex. **Q** is CAMAC Q and **X** is CAMAC X.

gen_cam command

The **gen_cam** command generates a command frame from station number **n**, sub-address **a**, function **f**, data **data** and a flag **flag** for the command frame, as follows;

```
1   % ./gen_cam
2   usage : ./gen_cam n a f data flag(1:start,2:end,0:normal,packet:others)
3   % ./gen_cam 4 0 16 0x555555 1
```

```
4   Data1 = 555555(hex)
5   Data2 = c0040010(hex)
```

The **Data1** and **Data2** corresponds to TxData1/RxData1 and TxData2/RxData2, respectively.

dec_cam command

The **dec_cam** command extracts station number **n**, sub-address **a**, function **f**, data **data** and status **status** from **Data1** and **Data2**.

```
1   % ./dec_cam
2   usage : ./dec_cam data1(hex) data2(hex)
3   % ./dec_cam 0x3555555 0xC0040010
4   n(4) a(0) f(16) data(0x555555) status(0x3)
```

put_cam command

```
1   % ./put_cam
2   usage : ./put_cam data(hex) cmd(hex)
3   ./put_cam 0x55555 0xc0040010
```

get_cam command

```
1   % ./get_cam
2   data = 0x55555 : reply = 0xC0040000
```

4.4.2 Check programs

exam0

The program **exam0** checks basic CAMAC operations in Programmed I/O. It includes Z, C, SetInhibit, RemoteInhibit, DisableInt, EnableInt, WriteEnableBit, ReadEnableBit, SetFastCycle, ResetFastCylce and Read/Write/NDT to a Switch register.

```
1   % ./exam0
2   number of reply frames : 13
3   ( 1) Z :                data( 4000000) =  4000000 rply(c0190011) = c0190011
4   ( 2) C :                data( 4000000) =  4000000 rply(80190010) = 80190010
5   ( 3) set Inhibit :      data( 4000000) =  4000000 rply(8019001a) = 8019001a
6   ( 4) remove Inhibit :   data(       0) =        0 rply(80190018) = 80190018
7   ( 5) disable interrupt : data(  ffffff) =   ffffff rply(80190118) = 80190118
8   ( 6) enable interrupt :  data(10ffffff) = 10ffffff rply(8019011a) = 8019011a
9   ( 7) write enable bits : data(10ffffff) = 10ffffff rply(80190110) = 80190110
10  ( 8) read enable bits :  data(10ffffff) = 10ffffff rply(8019011a) = 8019011a
```

```
11  ( 9) set fast cycle :    data(90000000) = 90000000 rply(8019021a) = 8019021a
12  (10) reset fast cyclc :  data(10000000) = 10000000 rply(80190218) = 80190218
13  (11) write data to SW :  data(13000000) = 13000000 rply(80040010) = 80040010
14  (12) read data from SW : data(13aaaaaa) = 13aaaaaa rply(80040000) = 80040000
15  (13) clear data in  SW : data(12000000) = 12000000 rply(a004000a) = a004000a
```

exam1

The program **exam1** executes CAMAC write/read operations in Programmed I/O and DMA to Switch register. The 4 types of processes are available. **cam_exec_pio** (default) for PIO. **cam_exec_dma**, **cam_exec_dma_seq** and **cam_exec** for DMA. There are 4 data patterns to be written/read/checked. One is a series of data pattern of alternative 0xFFFFFF and 0. Another is a series of data pattern of alternative 0xFFFFFF, 0, 0x555555, 0xAAAAAA and 0. Third one is a series of data 0, 1, 2, and so on. Final one is a series of random number. The values of the parameter **pattern** are 0, 1, 2 and 3 according to the order of above explanation. The default value is 1.

```
1   % ./exam1
2   usage : ./exam1 process:0 [pattern:1] [loop:1] [num_frame] [fast]
3           process 0: cam_exec_pio (default)
4           process 1: cam_exec_dma
5           process 2: cam_exec_dma_seq
6           process 3: cam_exec
7           pattern 0: a series of data (0xFFFFFF, 0)
8           pattern 1: a series of data (0xFFFFFF, 0, 0x555555, 0, 0xAAAAAA) (default)
9           pattern 2: a series of data (0,1,2,3,4,5...)
10          pattern 3: a series of random data
11          loop     : iteration count to be executed (default = 1)
12          num_frame: number of frame to be executed (default = 10)
13          fast     : fast cycle:1 normal cycle:0 (default = 0)
14  % ./exam1 0 2 10 100
15  Start CAMAC access to Switch register...
16  100 command frames...
17  Execution with loop = 10
18  exam1 has been done successfully
19  %
```

exam2

The program **exam2** checks basic DAQ operations in Programmed I/O. It includes read event count, clear busy-out, clear event counter, enable/disable trigger input, set pulse out and reset pulse out.

```
1   % ./exam2
2   number of reply frames : 8
3   read event count     : data(       0) =        0 rply(d0000000) = d0000000
4   clear busy out       : data(       0) =        0 rply(90010000) = 90010000
5   clear event counter  : data(       0) =        0 rply(90020000) = 90020000
6   enable trigger input : data(       0  =        0 rply(90030002) = 90030002
7   disable trigger input: data(       0) =        0 rply(90040000) = 90040000
8   set pulse out        : data(       0) =        0 rply(90050008) = 90050008
9   reset pulse out      : data(       0) =        0 rply(b0060000) = b0060000
```

exam3

The program **exam3** checks CAMAC LAM Interrupt operations in Programmed I/O. You should put NIM signal into LEMO connector of Interrupt register.

```
1   % ./exam3 10
2   Loop count = 1 : LAM pattern = 2
3   Loop count = 2 : LAM pattern = 2
4   Loop count = 3 : LAM pattern = 2
5   Loop count = 4 : LAM pattern = 2
6   Loop count = 5 : LAM pattern = 2
7   Loop count = 6 : LAM pattern = 2
8   Loop count = 7 : LAM pattern = 2
9   Loop count = 8 : LAM pattern = 2
10  Loop count = 9 : LAM pattern = 2
11  Loop count = 10 : LAM pattern = 2
```

exam4

The program **exam4** checks DAQ Trigger Interrupt operations in Programmed I/O. You should put NIM signal into LEMO connector of TRIG-IN at the CAMAC controller.

```
1   % ./exam4 10
2   Loop count = 1 : Event count = 1
3   Loop count = 2 : Event count = 2
4   Loop count = 3 : Event count = 3
5   Loop count = 4 : Event count = 4
6   Loop count = 5 : Event count = 5
7   Loop count = 6 : Event count = 6
8   Loop count = 7 : Event count = 7
9   Loop count = 8 : Event count = 8
10  Loop count = 9 : Event count = 9
11  Loop count = 10 : Event count = 10
```

4.5 Programming

The section introduces how to use the CAMAC library dedicated to pipeline CAMAC controller. The general purpose CAMAC library is traditionally used. It is easy to understand how to use the library. However, the dedicated library is a little bit complicated.

4.5.1 Command frame generation

First, CAMAC command frames are generated by using the command frame generator like this;

```
1    if( (status = cam_gen_init( length, cmdbuf )) == -1 ) {
2      printf(``cam_gen_init error...\n'');
3      exit(0);
4    }
5    if( (status = cam_gen_cc( cmdbuf, 25, 0, 17, 0 )) == -1 ) { // Z
6      printf(``cam_gen_cc error ...\n'');
7      exit(0);
8    }
```

cam_gen_init in the line 1 initializes a command buffer **cmdbuf**, which length is specified by **length**. **cam_gen_cc** in the line 5 generates a command frame from N(25), A(0), F(17) and Data(0). The frame is stored into the command buffer **cmdbuf**. The frame header is automatically generated. You can call another **cam_gen_cc** routine for a CAMAC command frame or DAQ command frame by calling **cam_gen_daq** routine. If you want to finish generating command frames, you can do it anytime. You do not care of the frame header while the generator automatically makes first frame header, end frame header and so on.

4.5.2 Command frame execution

After calling the command frame generator, a CAMAC open routine and CAMAC/DAQ execution routines such as for PIO routine should be called.

```
1    if((fd = cam_open( )) == -1) {
2      printf(``cam_open error\n'');
3      exit(0);
4    }
5    if( (status = cam_exec_pio( fd, cmdbuf, rplybuf )) < 0 ) {
6      printf(``cam_exec_pio error\n'');
7      exit(0);
8    }
```

cam_open opens CAMAC and then get a file descriptor. The file descriptor is used for other routines to identify the CAMAC port. **cam_exec_pio** executes the command frames in **cmdbuf** in PIO mode and then stores the reply frames into a reply frame buffer **rplybuf**. If you want to execute in DMA mode, **cam_exec_dma** or **cam_exec_dma_seq** should be called. The former starts read DMA first and then executes write DMA while the latter executes write DMA and then does read DMA next. The latter operation depends on the total FIFO size in PCI control unit, but the former does not.

The execution performance of PIO and DMA depends on the calling overhead and the execution speed. On one hand, the PIO operation has small overhead while the DMA operation has much overhead. On the other hand, the execution speed of the DMA is faster than that of the PIO. When the frame length is small, the PIO operation is better. When the frame length increases, the DMA operation is superior. The combine routine of the PIO and the DMA operations is provided. **cam_exec** is used for this purpose.

4.5.3 Data and status extractions

After the operation, you can extract CAMAC/DAQ data and the status from reply frames in reply frame buffer.

```
1       status = cam_extract_cc_data(rplybuf, len,
2                                    &actual_length, databuf );
3       status = cam_extract_cc_status(rplybuf, len,
4                                    &actual_length, statusbuf );
5       status = cam_extract_cc_qx( statusbuf[0], &q, &x );
```

cam_extract_cc_data in the line 1 copies data in **rplybuf** of length **len** into a data buffer **databuf**. The actual length of data is returned into **actual_length**. **cam_extract_cc_status** copies status in **rplybuf** of length **len** into a status buffer **statusbuf**. The status in the status buffer includes CAMAC Q and CAMAC X if the frame is CAMAC reply frame. Thus, you can extract the Q and the X by using **cam_extract_cc_qx**.

Finally, CAMAC close routine **cam_close** is called.

4.5.4 Interrupt handling

There are two type of interrupt handling. One is CAMAC LAM interrupt. Another is DAQ trigger interrupt.

CAMAC LAM interrupt

The following procedure is an example of handling CAMAC LAM interrupt. Clear interrupt at Interrupt register first in line 2. The function 9 is used. Next, enable interrupt at CAMAC controller in line 4. In the example, All CAMAC stations are enabled by specifying the mask value of 0xFFFFFF. Then, enable interrupt at the Interrupt register

in line 6 and wait for an interrupt from the register in line 8. Finally, disable interrupt at the Interrupt register in line 10. You may disable interrupt at the CAMAC controller, but it is not necessary in this example.

```
1    // clear interrupt at Interrupt register
2    status = cam_single_cc(fd, INTREG, 0, 9, &data, &q, &x);
3    // enable interrupt at the controller
4    status = cam_enable_lam(fd, 0xFFFFFF);
5    // enable interrupt at Interrupt register
6    status = cam_single_cc(fd, INTREG, 0, 26, &data, &q, &x);
7    // wait for a CAMAC LAM interrupt
8    status = cam_wait_lam( fd, &lam_pattern, TIMEOUT );
9    // disable interrupt at Interrupt register
10   status = cam_single_cc(fd, INTREG, 0, 24, &data, &q, &x);
```

DAQ trigger interrupt

The procedure for DAQ trigger interrupt is rather simple. Enable the interrupt first in line 2. Then, wait for an interrupt in line 4. After an interrupt occurs, disable the interrupt in line 6. Finally, clear the busy-out signal in line 8. Even if multiple triggers occur, any interrupt does not occur and event counter is not incremented before the busy-out signal is cleared.

```
1    // enable DAQ trigger interrupt
2    status = cam_enable_trig(fd);
3    // wait for an interrupt
4    status = cam_wait_trig( fd, &event_count, TIMEOUT );
5    //disalbe the interrupt
6    status = cam_disable_trig(fd);
7    // clear busy out
8    status = cam_single_daq( fd, DAQEXE_CTRL_CLRBSY, &data, &daq_status);
```

Chapter 5

Linux System

The choice of a suitable Operating System (OS) for PC compatible hardware is dependent on the expected functionality. Since the software tools, like compilers and utilities are, Open Source Software / GNU products, a commodity Open Source OS, Linux is chosen. A Linux distribution kit tailored by KEK is implemented for the pipeline CAMAC controller.

The board computer has a flash disk. The Linux system on the board computer can boot from the flash disk. However, the flash disk has a limited life expectancy. The life expectancy is measured in the number of erase cycles. Linux frequently erases the directories of /var and /tmp. Thus, they are implemented as RAM disk.

There are some application programs using CAMAC. One is a set of examples to use CAMAC device driver / the library. The examples show the usage of all subroutines in the library. Another is an example for Java programming using the library. The program shows the usage of the library for WEB application. The other is a simple remote access program written in C language. The program is useful to use the library on the board computer from a remote machine.

Finally, the recovery procedure is introduced. If the flash disk is erased by some reason, new Linux system should be installed on the flash disk. The procedure is described. There is a CD-ROM KNOPPIX installed. The CD-ROM includes not only KNOPPIX-based Linux system but also the flask disk-based Linux system including CAMAC utility. The PC run KNOPPIX system is necessary. The operation manual of the pipeline CAMAC controller[16] explains them in detail.

5.1 Linux Installation using KNOPPIX CD including CAMAC utility

The following hardwares are necessary.

- PC can run KNOPPIX 3.2 with USB
- USB compact flash reader/writer

- KNOPPIX 3.2 CD-ROM including CAMAC utility
- 512 MB flask disk

5.1.1 How to get the distribution kit

KNOPPIX 3.2 CD-ROM including CAMAC utility is supplied when the pipeline CAMAC controller is purchased. The image of the CD is also given on the URL[22].

5.1.2 How to install Linux system

After KNOPPIX boots, login as root.

```
1   # fdisk /dev/sda
2   command : m
3   command : p
4   command : d
5   command : p
6   command : n
7   region(1-4) : 1
8       (1-993) : 993
9   command : w
10  # mke2fs -j /dev/sda1
11  # mount -t ext3 /dev/sda1 /mnt/sda1
12  # cd /mnt/sda1
13  # tar xzf /KNOPPIX/TOYO/compact.tar.gz
14  # e2label /dev/sda1 /
15  # cd /mnt/sda1/sbin
16  # ./grub
17  # grub> root (hd1,0)
18  # grub> setup (hd1)
19  # grub> quit
10  #
```

The line 1-9 show the procedure to make a partition by using fdisk command. The mke2fs command on the line 10 makes a ext3 file system. The flask disk connected to USB is manipulated as a SCSI disk. The compact.tar.gz shown in the line 13 includes a flask disk-based Linux system with CAMAC utility. The procedure shown in the line 14-19 including e2label and grub commands writes boot block for self-booting.

5.2 Application Software

Two application programs are provided in CC/NET CAMAC distribution kit.

5.2.1 Simple remote access programs written in C language

There are two programs, a client program and a server program. The server program runs on CC/NET Linux system while the client program runs on any remote computer. The communication protocol is based on TCP/IP socket. It does not adopt any middle-ware such as CORBA and Java RMI. The communication library is written in C language. The server program calls any routines of CC/NET CAMAC library according to the request from the client program. The client program is as same as a CAMAC program running on CC/NET Linux system. After the CAMAC program can run successfully on CC/NET Linux system, the program can become the client program on any remote computer without the modification of the program.

Here is a sample program included in the check programs of CC/NET CAMAC distribution kit, called **cam**. The following **Makefile** makes the server program called **ccnet_server** and the client program called **cam**.

The following describes the software components.

- ccnet.h : An include file for ccnet
- ccnet_server.c : A main program of the server.
- dispatch.c : It calls CAMAC access routines in CC/NET CAMAC library for accessing the CAMAC and is used for the server program.
- clientlib.c : It emulates CC/NET CAMAC library for the client.
- message.c : A common message passing library. It is used for the server and client programs.

A host name and a port number used in the client program should be assigned in the **ccnet.h**. The following procedure is an example to run the server and client programs.

```
1   [server] % ./ccnet_server &
2
3   [client] % ./cam 4 0 16 0xffffff
4   [client] Q = 1 : X = 1
5   [client] % ./cam 4 0 0
6   [client] Q = 1 : X = 1 : data = ffffff
```

5.2.2 Remote Access program using Java Remote Method Invocation (JavaRMI)

The WEB application program can access CAMAC modules from a remote machine. The JavaRMI[23] enables remote CAMAC access via a WEB server (Apache) running on

pipeline CAMAC controller. The application is a simple example program, but you can extend the program easily for your own program.

When you build and run the program, the following components are additionally necessary, namely, A WEB server called Apache and Java2 SDK Standard Edition called J2SE including JavaRMI and JNI(Java Native Interface).

How to build

After installing Apache and the J2SE, you can check the Java version like this;

```
% java -version
java version ‘‘1.4.1_02’’
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_02-b06)
Java HotSpot(TM) Client VM (build 1.4.1_02-b06, mixed mode)
```

The following procedure is to make a serve-side program.

```
% cd /home/inoue/public_html/Web
% make clean
rm -f *.class cam.h libMyImpOfcam.so *.o core *~
% make
javac web3.java
javac cam.java
javah -jni cam
gcc -c camac.c
gcc -O -shared -I/usr/java/include -I/usr/java/include/linux \
cam.c camac.o -o libMyImpOfcam.so
javac ServerImpl.java
rmic ClientImpl
rmic ServerImpl
```

How to run

On the setup of Apache, the file /etc/httpd/conf/httpd.conf should be edited and then the line “ServerName onlsbc1.kek.jp:80” should be added for instance. Now let’s start the Apache.

```
% /sbin/service httpd start
```

Server programs rmiregistry for JavaRMI and ServerImpl for CAMAC also run on the pipeline CAMAC controller like this;

```
% export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
% rmiregistry &
% java ServerImpl &
```

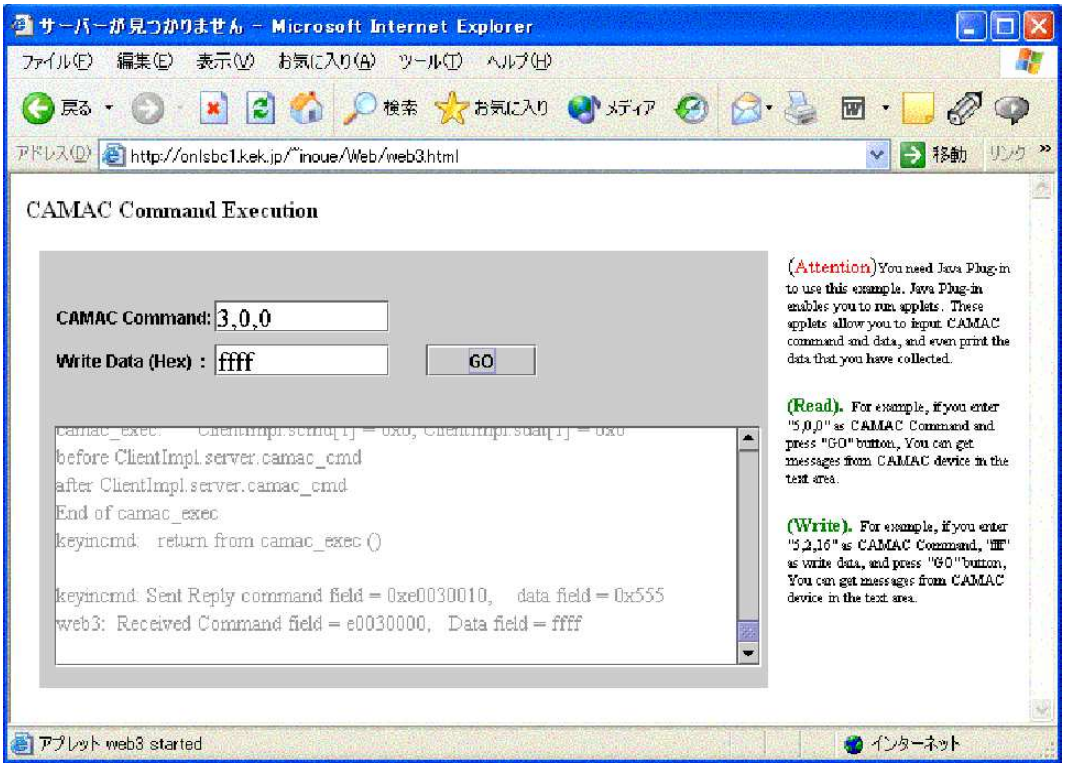

Figure 5.1: GUI for CAMAC

Figure 5.1 shows a GUI for CAMAC on Internet Explorer. When you want to do CAMAC read, you put N,A,F data such as "3,0,0" into CAMAC command box. You can get some messages including data in message box after clicking go button. For CAMAC write, N,A,F data and data to be written are put into the command box and Write data box, respectively.

Chapter 6

Performance

6.1 Environment and setup for the measurement

For the measurement, three types of computers are used. One is the pipeline CAMAC controller. It has Transmeta Crusoe 500 MHz processor. Another is a recent desktop PC. It has Intel(R) Xeon(TM) 2.80GHz processor. The other is a typical VME board computer. It has SPARC 300MHz processor. The detail of the specification is in Table 6.1. The operation system and the compiler are shown.

6.2 lmbench for Linux system performance measurement

lmbench is a micro-benchmark suite designed to focus attention on the basic building blocks of many common system applications, such as databases, simulations, software development, and networking.

lmbench provides a suite of benchmarks that attempt to measure the most commonly found performance bottlenecks in a wide range of system applications.

6.2.1 Timing issues

- Clock resolution: The benchmarks measure the elapsed time by reading the system clock via the gettimeofday interface. To compensate for the coarse clock resolution, the benchmarks are hand-tuned to measure many operations within a single time interval lasting for many clock ticks.

- Caching: If the benchmark expects the data to be in the cache, the benchmark is typically run several times; only the last result is recorded. If the benchmark does not want to measure cache performance it sets the size parameter larger than the cache. For example, the bcopy benchmark by default copies 8 megabytes to 8 megabytes, which largely defeats any second-level cache in use today.

Table 6.1: Environment and setup for the performance measurement

System	CPU	Operating System	C Compiler
CAMAC controller	Advantech PCM-9370, Transmeta Crusoe 500 MHz processor, TM5400 processor, 310MB main memory, 256 KB L2 cache memory	Linux kernel 2.4.18-27.8.0, based on Red Hat Linux 8.0	gcc version 3.2
Desktop PC	Intel(R) Xeon(TM) CPU 2.80GHz, 512MB main memory, 512 KB cache memory,	Linux kernel 2.4.18-27.7.x.cernsmp, based on CERN Red Hat Linux 7.3.2	gcc version 2.96
VME board computer	sun4u sparc SUNW,Ultra-5_10 (Force 50T) CPU 300MHz, 256MB main memory, 1MB cache memory	SunOS 5.8	gcc version 2.95.2

- Variability: The results of some benchmarks, most notably the context switch benchmark, had a tendency to vary quite a bit, up to 30%. It is suspected that the operating system is not using the same set of physical pages each time a process is created and thus the effects of collisions in the external caches is observed. It is compensated by running the benchmark in a loop and taking the minimum result.

6.2.2 Latency measurements

Operating system entry

Entry into the operating system is required for many system facilities.

Figure 6.1 shows latency of getppid system call. The getppid system call gets the process ID of the parent of the current process. Figure 6.2 shows latency of read system call to a null device.

Figure 6.3 shows latency of pipe system call. For measuring the latency, it uses two processes communicating through a Unix pipe to measure interprocess communication latencies. The benchmark passes a token back and forth between the two processes. No other work is done in the processes.

Table 6.2 shows the summary for those system calls.

Process creation costs

Process benchmarks are used to measure the basic process primitives, such as creating a new process. **lmbench** measures simple process creation by creating a process and

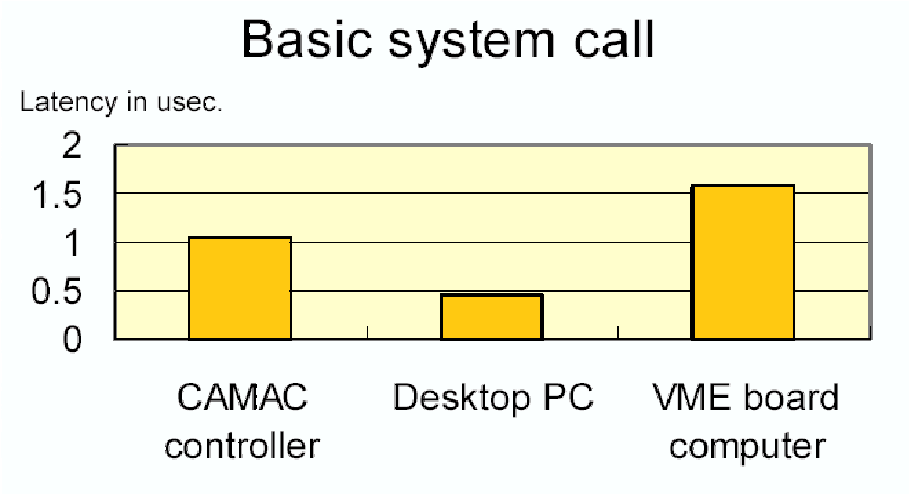

Figure 6.1: Latency of getppid system call

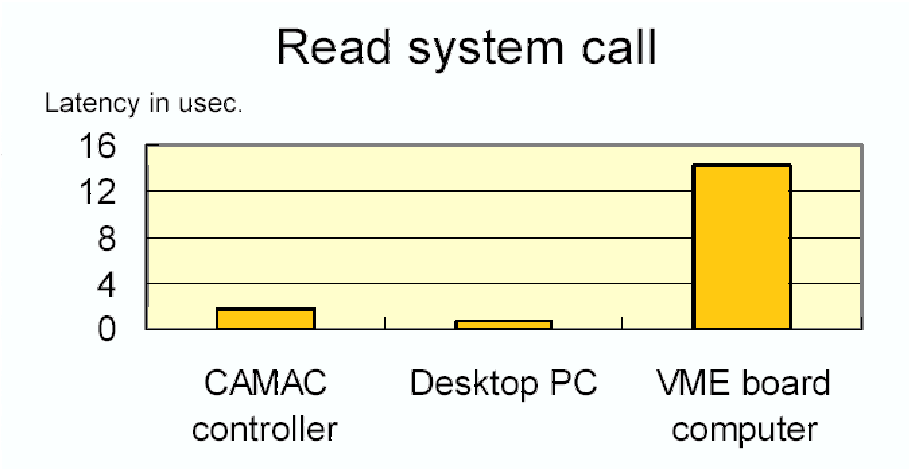

Figure 6.2: Latency of read system call

Table 6.2: Null system call time (in usec)

System	**getppid**	**read**	**pipe**
CAMAC controller	1.0372	1.7148	15.9827
Desktop PC	0.4586	0.7234	6.3920
VME board computer	1.5938	14.2222	46.8666

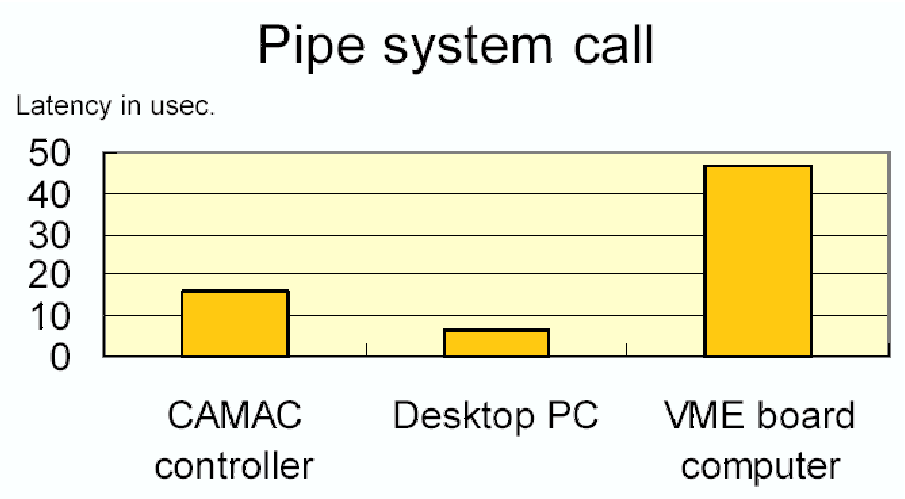

Figure 6.3: Latency of pipe system call

Table 6.3: Process creation time (in usec)

System	fork / exit
CAMAC controller	620.5000
Desktop PC	183.8333
VME board computer	3081.0000

immediately exiting the child process. The parent process waits for the child process to exit. The benchmark is intended to measure the overhead for creating a new thread of control, so it includes the fork and the exit time.

Figure 6.4 and Table 6.3 show latency of fork&exit system call.

6.2.3 Context switching performance

Context switch time is defined here as the time needed to save the state of one process and restore the state of another process. Context switches are frequently in the critical performance path of DAQ applications.

Typical context switch benchmarks measure just the minimal context switch time - the time to switch between two processes that are doing nothing but context switching. However, it depends on number of processes and size of processes.

- Number of processes : The context switch was measured as a ring of 2 to 96 processes that are connected with Unix pipes. A token is passed from process to process, forcing context switches. Each transfer of the token has two costs: the context switch, and the overhead of passing the token. In order to calculate just the context switching time, the benchmark first measures the cost of passing the token through

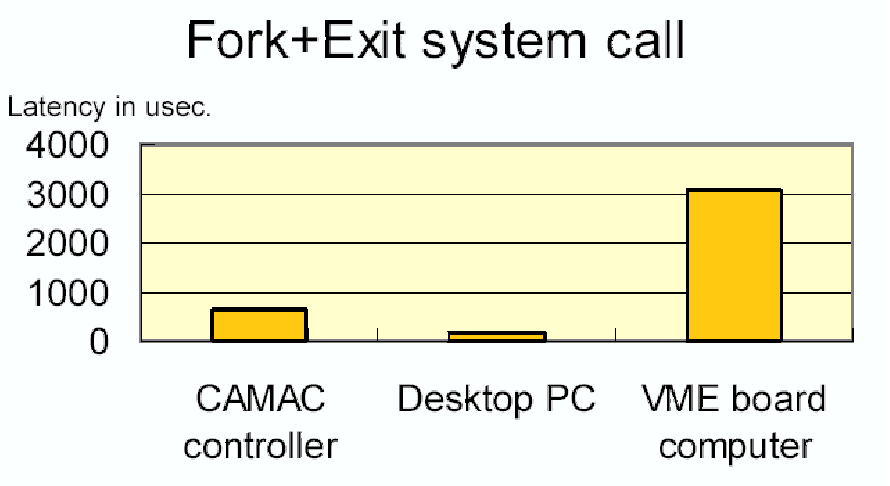

Figure 6.4: Latency of fork&exit system call

Table 6.4: Context switch times with 4 processes (in usec)

System	**4 KB**	**16 KB**	**64 KB**
CAMAC controller	5.31	9.22	91.84
Desktop PC	2.16	2.58	6.67
VME board computer	42.12	101.19	343.65

a ring of pipes in a single process. This overhead time is defined as the cost of passing the token and is not included in the reported context switch time.

- Size of processes : The context switch time depends on not only the number of processes but also the size of processes because it is dependent on the cache size of computers. Three sizes of processes are measured, namely, 4KB, 16KB and 64KB. When increasing the size, the cache overflow occurs.

Table 6.4 is a result of Context switch times with 4 processes.

Figure 6.5 shows process switching performance with 4&96 processes which size are 4KB.

Figure 6.6 shows process switching performance with 4&96 processes which size are 16KB.

Figure 6.7 shows process switching performance with 4&96 processes which size are 64KB.

Table 6.5 is a result of Context switch times with 96 processes.

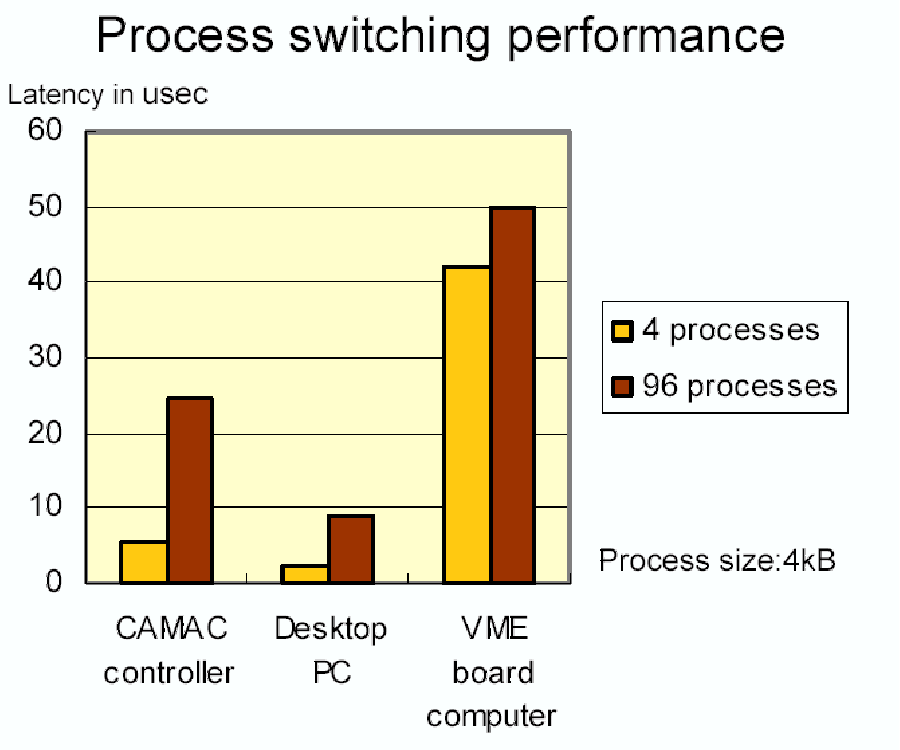

Figure 6.5: Process switching performance with 4&96 processes which size are 4KB

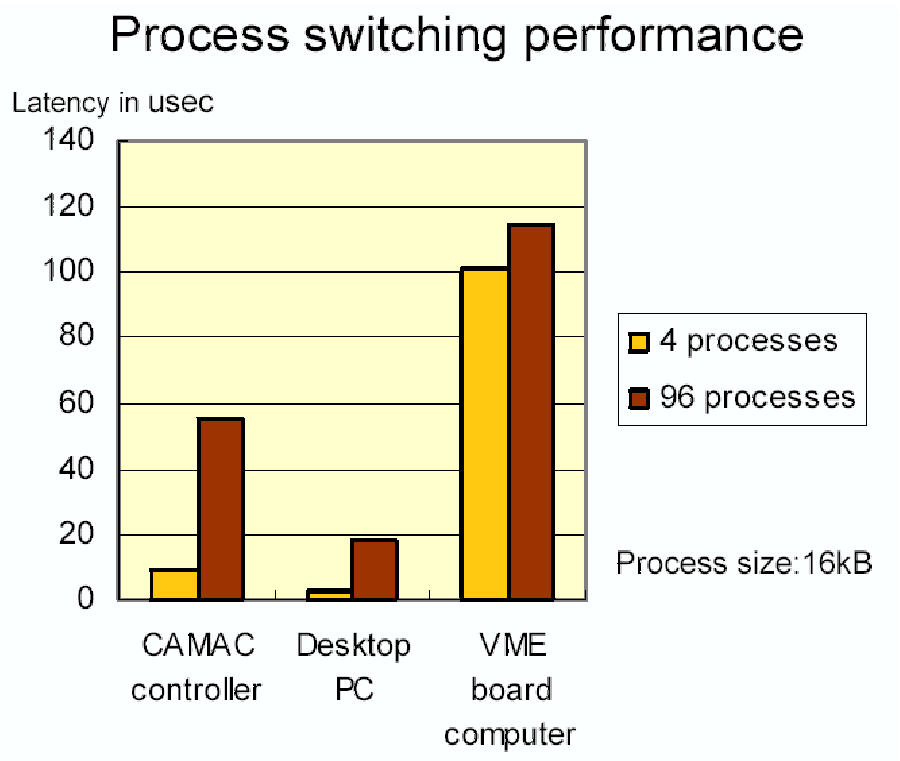

Figure 6.6: Process switching performance with 4&96 processes which size are 16KB

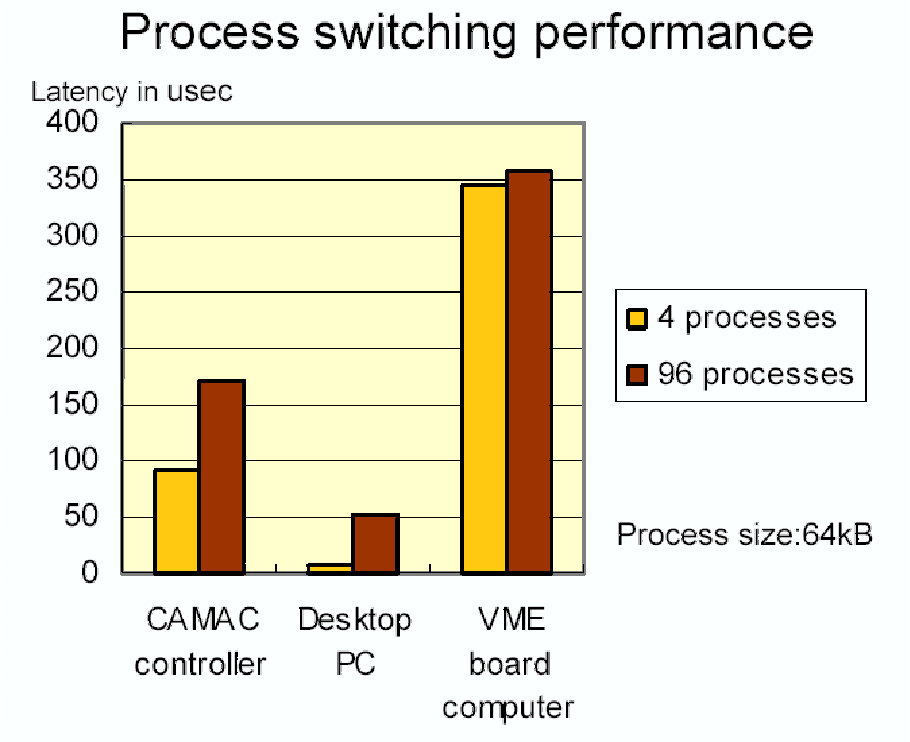

Figure 6.7: Process switching performance with 4&96 processes which size are 64KB

Table 6.5: Context switch times with 96 processes (in usec)

System	4 KB	16 KB	64 KB
CAMAC controller	24.74	55.06	169.93
Desktop PC	8.95	18.07	51.48
VME board computer	50.00	114.98	357.74

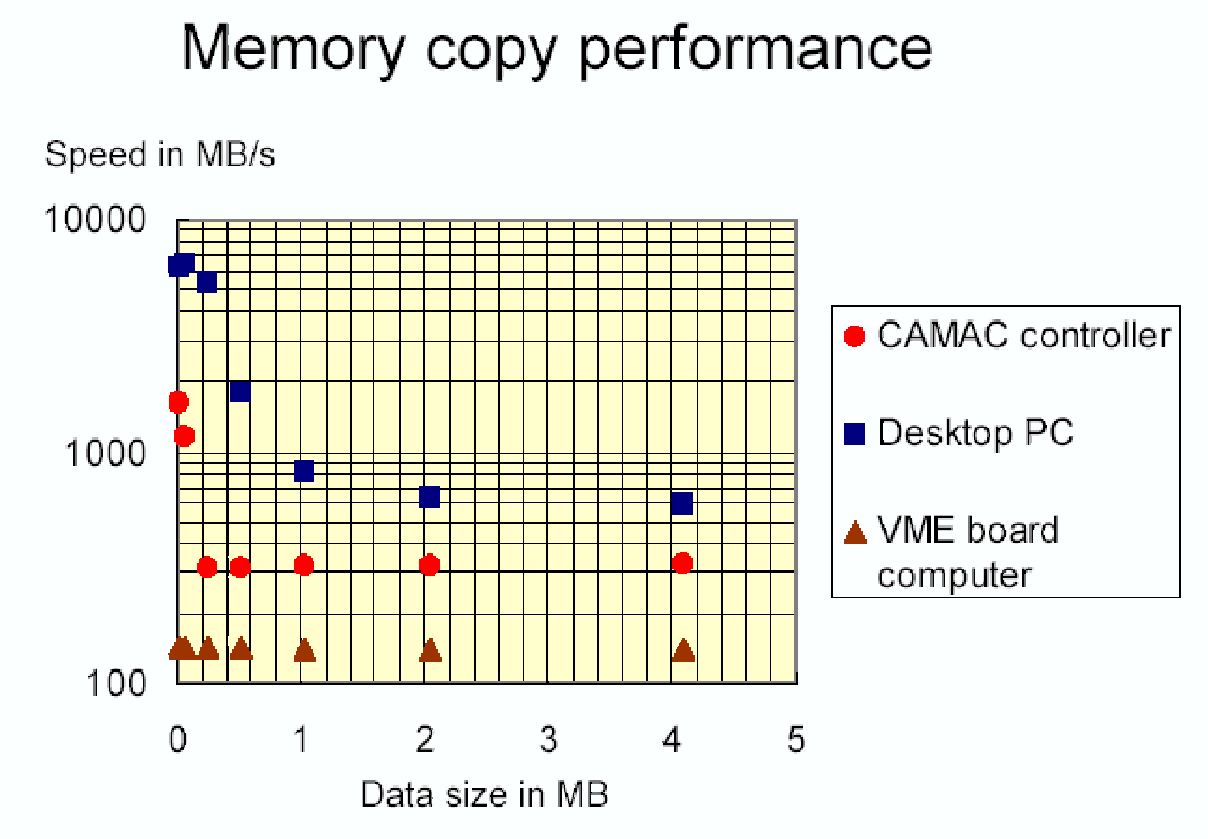

Figure 6.8: Memory Copy Performance

6.3 Other benchmark programs for Linux system performance measurement

6.3.1 Memory Copy Performance

This is a simple program to measure the memory copy performance using a system call "memcpy". The memory size varies from 1KB to 4 MB.

Figure 6.8 is a result of memory copy performance measurement.

6.3.2 NETPERF

Netperf[25] is a benchmark that can be used to measure the performance of many different types of networking. TCP throughput performance and TCP request & response performance are measured. The former shows the bandwidth the network and the latter shows the latency of the network. Figure 6.9 shows TCP throughput performance measured by using netperf utility.

Figure 6.10 shows TCP request & response performance measured by using netperf utility.

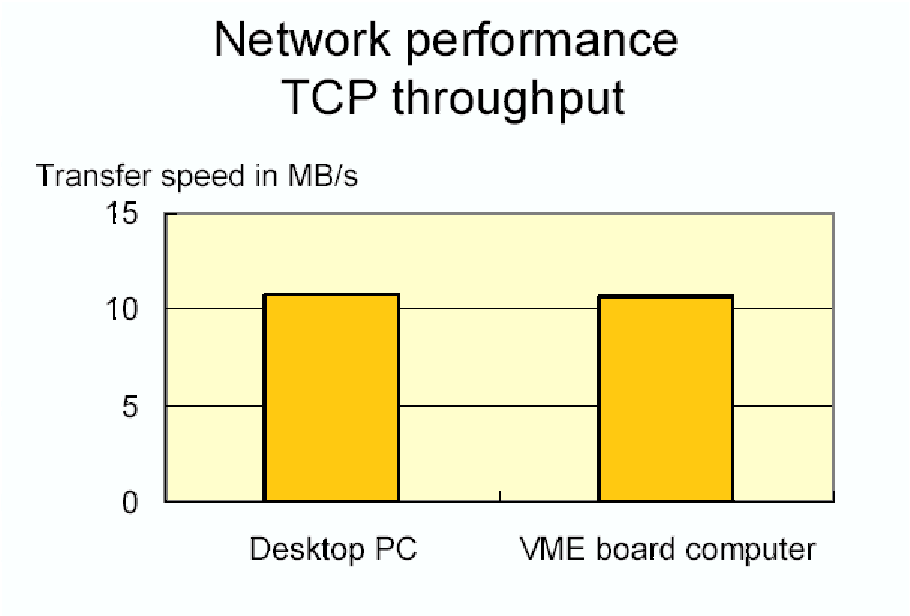

Figure 6.9: TCP throughput

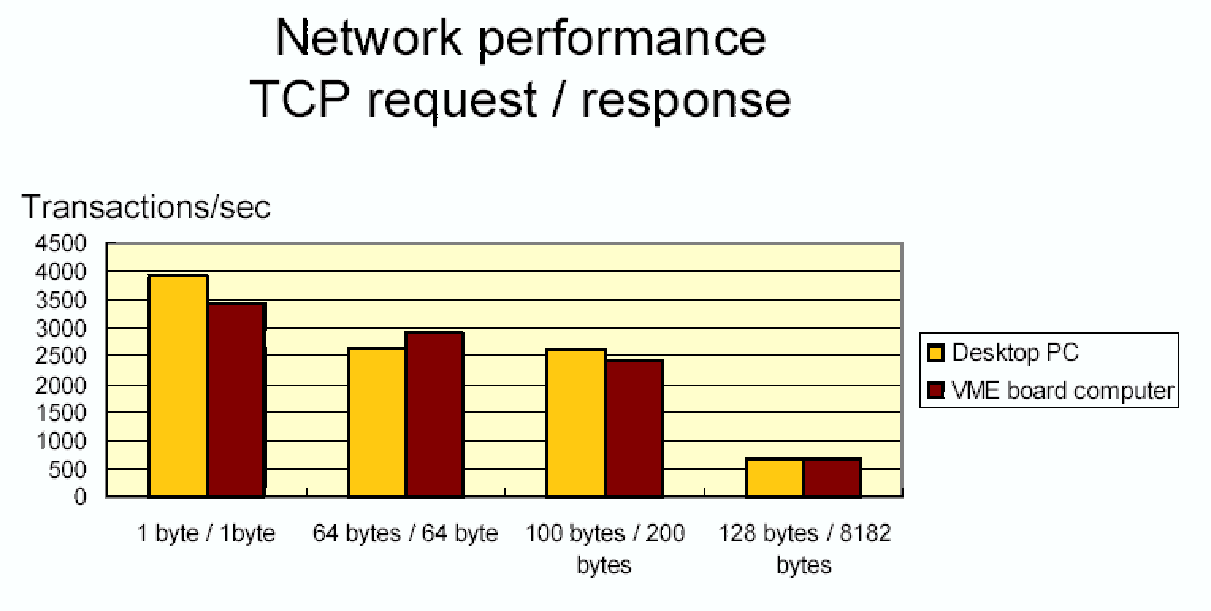

Figure 6.10: TCP request & response

Table 6.6: I/O port access performance

inl()	outl()
0.42 usec	0.45 usec

Table 6.7: performance of kernel routines

ioctl system call	copy_from_user	copy_to_user
1.29 usec	0.12 usec	0.13 usec

6.4 CAMAC performance

6.4.1 Performance of basic functions

The execution time of I/O port operation functions called inl() and outl() was measured. Table 6.6 shows the result of the measurement.

The overhead of ioctl system call was measured by using the CAMAC device driver. The memory copy performance was previously shown, but copy function of kernel routine in the device driver takes some overhead. There are two kernel routines, copy_from_user and copy_to_user. Those overheads were also measured. Actually, four bytes data are copied by using copy_from_user and copy_to_user.

From hardware point of view, the CAMAC executor executes a CAMAC normal command frame in 1.04 usec and a CAMAC packet command frame in 1.24 usec. When a packet includes only a CAMAC frame, it is called a packet frame. Usually a packet consists of multiple frame. The start frame follows normal frames and then a end frame at end. According to CAMAC protocol, CAMAC S2 timing signal should not be ignored, but some CAMAC modules works even if the signal is ignored. Fast mode ignores the timing signal. In fast mode, the executor executes the normal frame in 0.72 usec and the packet frame in 0.92 usec. Table 6.8 summaries those values.

Figure 1.2 shows the actual CAMAC performance measured at a CAMAC crate.

Table 6.8: CAMAC performance

frame	normal mode	fast mode
normal frame	1.04 usec	0.72 usec
packet frame	1.24 usec	0.92 usec

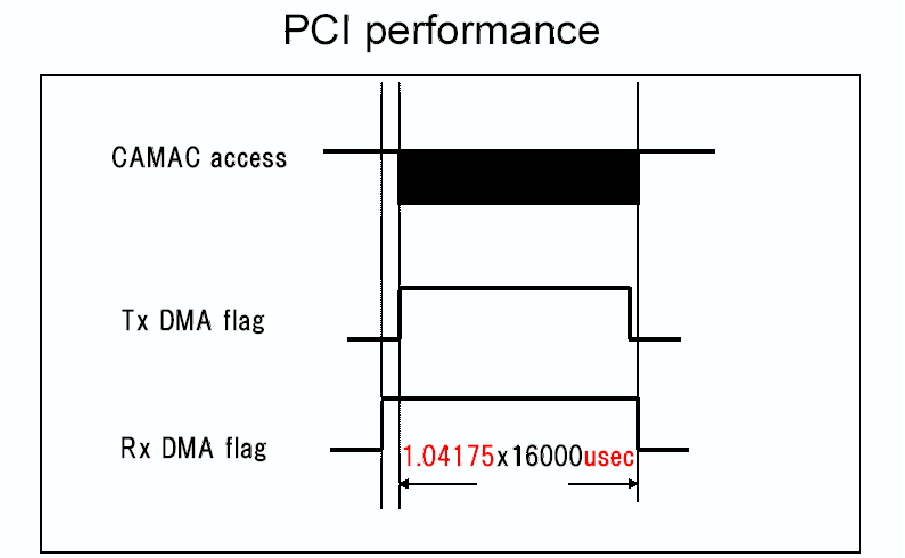

Figure 6.11: Block transfer performance

6.4.2 Block transfer performance

If we want to get best performance on the CAMAC operation, block transfer should be used. Figure 6.11 shows block transfer performance. In the Figure, Rx DMA flag is asserted first. This means CAMAC DMA read is done first while no actual read operation is done because CAMAC reply frames are not generated. When CAMAC DMA write operation starts, CAMAC command frames come into CAMAC executor and the it executes the frames. In the Figure, CAMAC access shows the execution. After the DMA write operation, it finishes before the completion of CAMAC operation. When the CAMAC operation is done, DMA read is also done. On the measurement, 16000 command frames are executed in 16,668 usec. This means a command frame is executed in 1.04175 usec.

6.4.3 Interrupt frame performance

There is a command frame handling an interrupt called TRIG signal. When the frame is executed at CAMAC executor, it waits for the signal input. While no signal, no CAMAC operation is done. When an interrupt occurs, it sends the frame to PCI control unit as the reply frame. Figure 6.12 shows interrupt frame performance. Input signal occurs at 100 kHz. The interrupt frame follows 4 command frames for CAMAC read. When an interrupt occurs, the frame is executed in 400 nsec and then the read CAMAC functions are executed in 1.04 x 4 usec. The figure shows 100 kHz readout with 4 CAMAC operation should be done successfully.

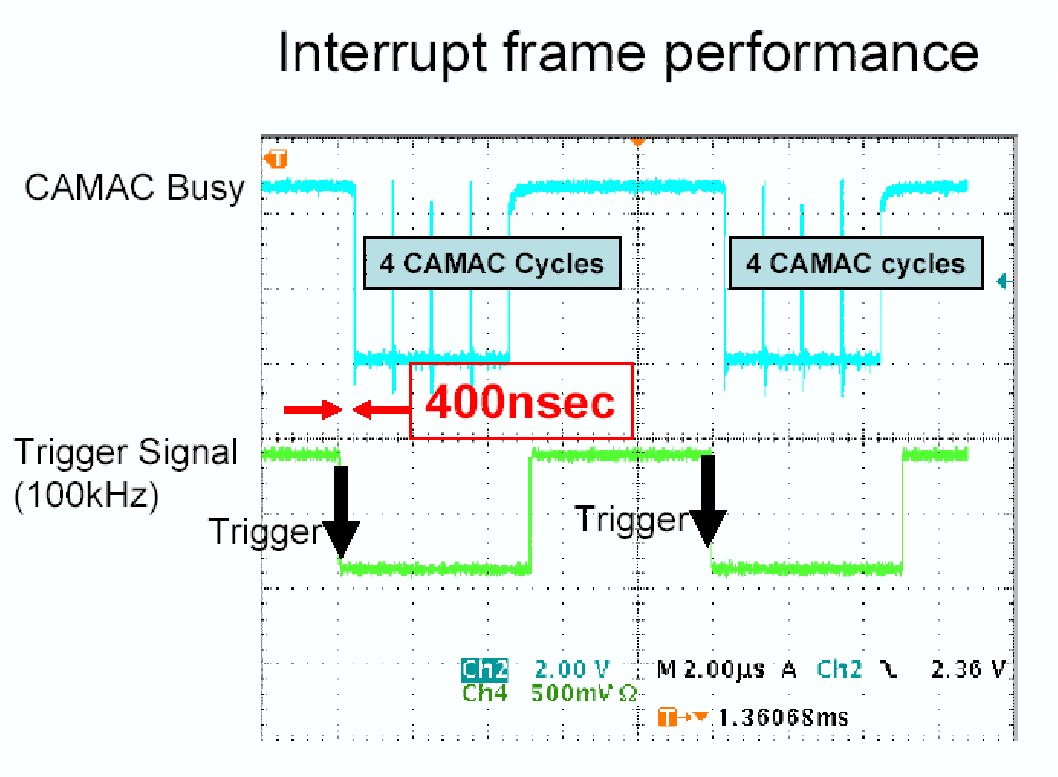

Figure 6.12: Interrupt Frame Performance

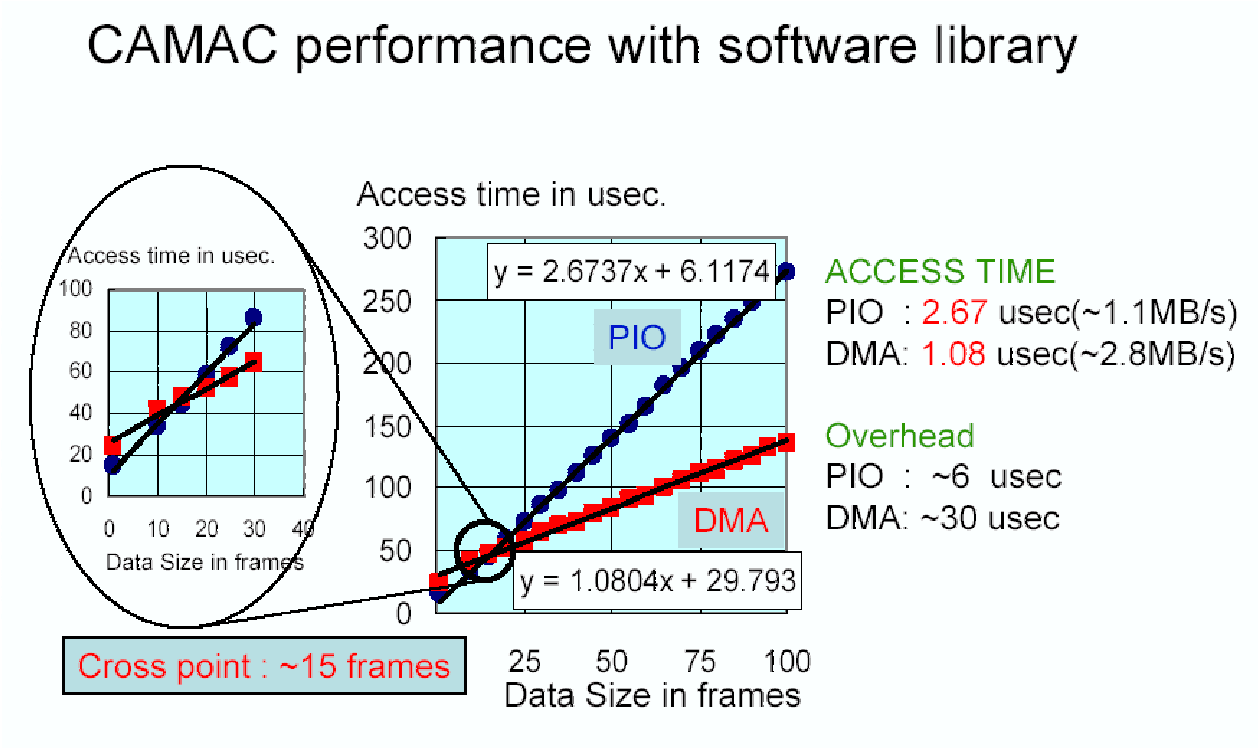

Figure 6.13: CAMAC performance with software library

6.4.4 CAMAC performance with the CAMAC library

When CAMAC users access the pipeline CAMAC controller, they will use the CAMAC software library. Actual performance of the controller with the software library should be measured. Figure 6.13 shows CAMAC performance with the software library. This is the preliminary result. There are two types of CAMAC operation by using the library. One is based on PIO while another is based on DMA. Access overhead of PIO is less than that of DMA. Thus, PIO is better than DMA in small words of CAMAC operation. The cross point is about 15 CAMAC frames. CAMAC operation in PIO takes 2.67 usec while that in DMA takes 1.08 usec.

6.5 Application performance

6.5.1 CAMAC remote access program

The remote access program executes CAMAC operations over network as if it executes on local node. It provides CAMAC library for client and a server program. CAMAC user can run their own CAMAC program which runs on local node, over network, without any modification. On the measurement, The desktop PC is used as the client node while the server run on the pipeline CAMAC controller itself. The performance of DMA operation is only measured. The results are also preliminary. Figure 6.14 shows CAMAC performance over network.

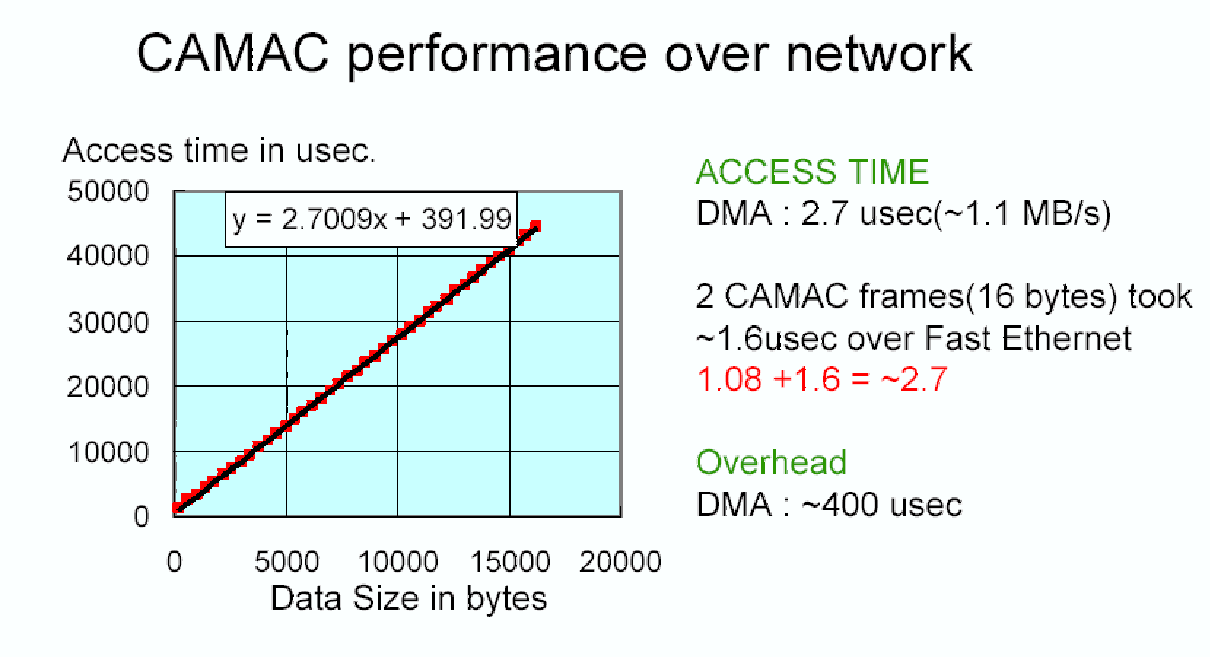

Figure 6.14: CAMAC performance over network

The access time per CAMAC word takes 2.7 usec. A CAMAC command frame consists of 8 bytes while the reply frame also does 8 bytes. This means 16 bytes of data passes over network. The overhead is 1.6 usec on network with 10MB/s bandwidth, which is real value measured on the two nodes. The overhead of DMA per CAMAC command/reply frame is 1.08 usec. The network overhead plus DMA overhead equals the access time per CAMAC word over network.

Bibliography

[1] IEEE, "IEEE Standard Modular Instrumentation and Digital Interface System (CAMAC) (Computer Automated Measurement and Control) IEEE Std 583-1975

[2] PC/104 Embedded Consortium, PC/104-Plus Specification Version 1.2, August 2001 http://www.pc104.org/

[3] Altera Corporation, http://www.altera.com/

[4] Debian GNU/Linux, http://www.jp.debian.org/

[5] Red Hat Linux distribution, http://www.jp.redhat.com/download/

[6] Home page of Parallel CAMAC Project, http://www-online.kek.jp/~ yasu/Parallel-CAMAC/

[7] Yet another Parallel CAMAC Project page (in Japanese), http://www-online.kek.jp/~ inoue/Parallel-CAMAC/

[8] TOYO Corp., CAMAC Crate Controller type CC/7700 and Host Interface Board types CC/ISA and CC/PCI in Japanese, http://www.toyo.co.jp/daq/index.html

[9] High Energy Accelerator Research Organization (KEK), http://www.kek.jp/

[10] TOYO Corporation, http://www.toyo.co.jp/

[11] Fird Corporation, http://www2.ocn.ne.jp/~ fird/

[12] KineticSystems Corp., VME to 3922 Interface w/DMA model 2917-Z1A & Model 2915-Z1A & PCI Interface to 3922 Model 2915-Z1A & Parallel Bus Crate Controller Model 3922-Z1B, http://www.kscorp.com/www/products/camac.htm

[13] Y. Yasu, E. Inoue, S. Harada and H. Kyoo, User Guide of Pipeline CAMAC Controller with PC104Plus Single Board Computer (this paper), http://www-online.kek.jp/~ yasu/Parallel-CAMAC/UserGuide/UserGuide.pdf

[14] A general purpose interconnect developed for Parallel CAMAC project.

[15] Advantech Co., Ltd., Board computer PCM-9370
http://www.advantech.co.jp/epc/SBC/biscuit/pcm9370.html

[16] E. Inoue, SBC Operation Manual,
http://www-online.kek.jp/~ inoue/Parallel-CAMAC/Work/OP-Man.htm

[17] E. Inoue, CAMAC page for Solaris operating system
http://www-online.kek.jp/~ inoue/CAMAC/

[18] K. Nakayoshi, USB-CAMAC Library for Linux
http://www-online.kek.jp/~ nakayosi/USB/uguide3929/uguide.html

[19] Welcome to Linux Home Page at KEK Online Group (old pages)
http://www-online.kek.jp/~ online/Linux/linux.html

[20] Y. Yasu, VME&CAMAC Device Drivers for HP-RT
http://onlhpux.kek.jp/hprt-drivers.html

[21] Y. Yasu, Usage Guide of CAMAC Library for UNIX
http://www-online.kek.jp/~ inoue/CAMAC-Doc/doc/users_guide_v1.pdf

[22] KNOPPIX, http://www.knoppix.org/, http://unit.aist.go.jp/it/knoppix/

[23] Java 2 Platform Standard Edition, http://java.sun.com/j2se/

[24] Larry McVoy (Silicon Graphics, Inc.) and Carl Staelin (Hewlett-Packard Laboratories), lmbench: Portable tools for performance analysis
http://www.bitmover.com/lmbench/

[25] Public Netperf Homepage
http://www.netperf.org/netperf/NetperfPage.html