CBD 8210

CAMAC Branch Driver

User's Manual, version 2.0

Designation: DOC 8210/UM

PN: 085.180

Version 2.0 - July 1996

CREATIVE ELECTRONIC SYSTEMS S.A.

Warranty Information

The information in this document has been checked carefully and is thought to be entirely reliable. However, no responsibility is assumed in case of inaccuracies. Furthermore, CES reserves the right to change any of the products described herein to improve reliability, function or design. CES neither assumes any liability arising out of the application or use of any product or circuit described herein nor conveys any licence under its patent rights or the rights of others.

WARNING

THIS EQUIPMENT GENERATES, USES AND CAN RADIATE RADIO FREQUENCY ENERGY AND MAY CAUSE INTERFERENCE TO RADIO COMMUNICATIONS IF NOT INSTALLED AND USED IN ACCORDANCE WITH THE INSTRUCTION MANUAL. IT HAS BEEN TESTED AND FOUND TO COMPLY WITH THE LIMITS OF A CLASS A COMPUTING DEVICE PURSUANT TO SUB-PART J OF PART 15 OF FCC RULES, WHICH ARE DESIGNED TO PROVIDE REASONABLE PROTECTION AGAINST SUCH INTERFERENCES WHEN OPERATED IN A COMMERCIAL ENVIRONMENT. OPERATION OF THIS EQUIPMENT IN A RESIDENTIAL AREA IS LIKELY TO CAUSE INTERFERENCE; IN WHICH CASE, THE USER AT HIS OWN EXPENSE WILL BE REQUIRED TO TAKE WHATEVER MEASURES ARE NECESSARY TO CORRECT THE INTERFERENCE.

Creative Electronic Systems
70, Route du Pont-Butin
P.O. Box 107
CH-1213 PETIT-LANCY 1
SWITZERLAND

CONTENTS

1. INTRODUCTION

1.1 Overview

This Module, type CBD 8210, is a double height VME card allowing a CAMAC Branch (EUR 4600) to be driven from VME. This means that up to 7 CAMAC crates, on the same branch highway, can be accessed from VME. The use of PALs and I.C.s in the mechanical SO format makes it a high performance module. The design of this module has been made with the intention of allowing it to work in conjunction with a second module enabling DMA transfers to be made from CAMAC to VME.

The module is an update of the CAMAC Branch Driving VME module developed by the French Research Institute at SACLAY. The characteristics are the following:

a) Programmable 16-bit or 24-bit transfers.
b) Two external interrupts with handshake.
c) Comprehensive LAM handling.
d) Addition of a DMA module at a later date.
e) Complete monitoring of all transfers displayed on the front panel.
f) Multi-crate addressing possible.

1.2 Address Mapping

The mapping of the CAMAC address field has been taken from the CERN publication "CERN IMPLEMENTATION RECOMMENDATION for MC 68000 BASED CAMAC PORT CONTROLLERS". Therefore, for the 24-bit address of VME the following bit allocation exists:

bits <23...22>	1…0
bits <21...19>	Branch address (0 to 7 = 8 branches)
bits <18...16>	Crate address 1 - 7 standard addressing.
bits <15...11>	N address - CAMAC station number
bits <10...07>	A address - CAMAC sub-address
bits <06...02>	F code - CAMAC function
bit <01>	CAMAC Word length 0 = 24-bit 1 = 16-bit

Thus all the CAMAC system parameters are mapped onto the VME address field.

The CBD 8210 can only drive one CAMAC branch where the number of the branch to be driven is selected by a front panel switch.

In practice, when a cycle of 24 bits is to be carried out, the master effects a first cycle with AD01 = 0 (detection of a 24-bit cycle) followed by a cycle with AD01 = 1 allowing the utilization of the instructions "LWORD" of the MC 68000.

1.3 Selection of Internal Registers

The internal registers are selected by using the command CR0 N29. They are as follows:

CSR	CR0 N29 A0 F0	Read / Write	
IFR	CR0 N29 A0 F4	Write	
INT. CONT.1	CR0 N29 A0 F5	Read / Write	Control
INT. CONT.1	CR0 N29 A0 F1	Read / Write	Data
INT. CONT.2	CR0 N29 A0 F6	Read / Write	
INT. CONT.2	CR0 N29 A0 F2	Read / Write	
INT. CONT.3	CR0 N29 A0 F7	Read / Write	
INT. CONT.3	CR0 N29 A0 F3	Read / Write	
CAR	CR0 N29 A0 F8	Read / Write	
BTB	CR0 N29 A0 F9	Read	
BZ	CR0 N29 A0 F9	Write	
GL	CR0 N29 A0 F10	Read	

2. INTERNAL REGISTERS

2.1 Control & Status Register - CSR

This Read / Write register contains all the information necessary to enable the correct functioning of the CBD 8210. It is formatted as follows:

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Q	X	TO	BD	MNOX	SY5	SY4	SY3	SY2	SY1	MTO	MLAM	MIT2	MIT4	IT2	IT4

INIT = Power-up + SYSRESET

Definition of Bit Allocation:

bit <15>	Q	Status of Q during the last CAMAC cycle. Read only.
bit <14>	X	Status of X during the last CAMAC cycle. Read only.
bit <13>	TO	Time-Out status flag of the last CAMAC cycle. Read only.
bit <12>	BD	CAMAC branch demand - transparent read of branch highway. Read only.
bit <11>	MNOX	no X mask. Allows masking of the BERR when the CAMAC cycle with X = 0. MNOX = 1 NOX gives DTACK MNOX = 0 NOX gives BERR Read / Write INIT = 1
bits <10...06>	SY5..SY1	Identification SY1 and SY2 monitored on the front panel. Read / Write INIT = 0
bit <05>	MTO	Time-Out mask. Allows removal of internal Time-Out. MTO = 1 No Time-Out on board. MTO = 0 Time-Out active. Read / Write INIT = 1
bit <04>	MLAM	Interrupt mask for level 3 VME of the control of GLAM. It suppresses also the automatic scanning of BG on BD = 1. Read / Write INIT = 1
bits <03...02>	MIT2,4	Mask for the external interrupts. Read / Write INIT = 1
bits <01...00>	IT2,4	External interrupt flags give O/P on ACK Lemos. Read only INIT = 0 The bits <00...01> - IT2, IT4 - are read only in the CSR register in order to prevent the erasure of interrupts.

2.2 Interrupt Flag Register - IFR

This register is write only and allows to set or reset the external interrupts by software. See chapter on external interrupts.

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
-	-	-	-	-	-	-	-	-	-	-	-	-	-	IT2	IT1

2.3 Interrupt Controller Registers - ICR

These registers correspond to the internal registers of the AMD 9519. Each interrupt controller consists of two registers - one for DATA and one for CONTROL.

For a more detailed explanation see the section on LAM handling and the data sheet on the 9519A.

2.4 Crate Address Register - CAR

This register is used for multiple addressing of the crates on the CAMAC branch and allows selection of the crates required for this action.

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Do not care								CR7	CR6	CR5	CR4	CR3	CR2	CR1	-

INIT = ?

Read / Write.

Note The CAR must not be READ when GL Scanning is ACTIVE.

2.5 BTB Registers - BTB

This register contains all the information regarding which crates in the branch are ON LINE. In addition, it removes the ambiguity of Time-Out.

This register is also used to generate the BG cycles and is "Read only".

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Do not care								BTB7	BTB6	BTB5	BTB4	BTB3	BTB2	BTB1	0

2.6 BZ Generation

The generation of BZ (reset CAMAC branch) is done by a write cycle to the address of the BTB register. The signal BZ is active for a period of approximately 15 µs.

BZ can also be generated by the front panel push button.

2.7 GL Register

In order to facilitate the use of this module the possibility to carry out BG cycles under program control has been included. During a read of this register the CBD 8210 carries out a standard CAMAC branch cycle using the BTB register to select the ON LINE crates. The logic of the 24-bit word is identical to the read CAMAC cycles.

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
GL16	GL15	GL14	GL13	GL12	GL11	GL10	GL09	GL08	GL07	GL06	GL05	GL04	GL03	GL02	GL01

T=1

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	0	0	0	0	GL24	GL23	GL22	GL21	GL20	GL19	GL18	GL17

T=0

2.8 Address Modifier

Decoding of the address modifier is carried out by means of a PROM 24S10 and is therefore user programmable. (See explanatory note).

To reprogram the address modifier decoder:

A0 = AM0

A1 = AM1

A2 = AM2

A3 = AM3

A4 = AM4

A5 = AM5

A6 = 0

A7 = 0

Qo = AMOK

Standard VME AM decoded : 3D, 39.

3. CAMAC TRANSFERS

3.1 Types of Transfer

The three types of transfer are separated inside the card. They are as follows:

READ function	F0 to F7
WRITE function	F16 to F23
COMMAND functions	F8 to F15 & F24 to F31
READ functions are executed by	MOV (CAMAC), (EA) cycle VME.
WRITE functions are executed by	MOV (EA), (CAMAC) cycle VME.
COMMAND functions are executed by	MOV (CAMAC), (EA) cycle VME
	TST (CAMAC) cycle VME

For COMMAND functions the value of the CSR is transmitted during a Read cycle.

For an incorrect CAMAC cycle, e.g. VME Write with F0 A0, the CBD 8210 completes the cycle with DTACK but the CAMAC cycle is not executed.

3.2 Control and Execution of CAMAC Cycles

When a correct CAMAC cycle (Read / Write) is sent from VME, the decoding logic sends a CAMAC REQUEST to the CAMAC sequencer. From this moment the sequencer starts to get access to the branch by:

❶ Generating BCR, BN, BF, (DATA) BA

❷ Generating BTA.

❸ Waiting until all the crates addressed generate their BTBs.

❹ Confirming that VME cycle can complete.

❺ Completing the CAMAC cycle.

The CBD 8210 finishes the VME handshake at the moment when all the BTBs are in a "1" state (called S1 in the CBD 8210). The logic of the beginning and the end of the cycle is included for multi-crate addressing.

The DATA are separated for Read and Write to eliminate overlap and not to stretch the length of the VME bus cycles.

In addition, the PAL-sequencer, controlling the CAMAC branch cycle, is protected against corruption in the event of a new branch cycle being requested before the end of the current cycle.

Multi-addressing is executed with the crate address code CR0. In such a case the crates addressed are those currently stored in the CAR.

For 16-bit CAMAC transfers (both Read and Write) the data are transmitted in a pseudo-transparent manner. The variables CR, N, F, A are stored at the start of each CAMAC cycle.

3.3 CAMAC Status Information

The CAMAC status bits Q, X and TO are updated during each CAMAC cycle. At the beginning of each cycle, the internal memory for these bits is reset. The values of Q and X are memorized at the moment of the internal S1 (all BTBs at 1). The generation of BERR due to the absence of X depends on the mask MNOX (MNOX = 1 cause no BERR if X = 0). In all other cases the CBD 8210 replies with DTACK.

3.4 CAMAC Time-Out

For each CAMAC cycle a Time-Out circuit is triggered IF the mask MTO=0. In the event that the circuit timing is out before completion of the current CAMAC cycle the VME handshake is completed with BERR and the CAMAC branch cycle is aborted with the TO flip flop being set. The duration of the Time-Out circuit can be selected by jumpers. The value selected can be between 2 µs and 134 s. See section on jumper settings.

3.5 24-bit CAMAC Transfers

24-bit CAMAC transfers are treated in the same way as with 32-bit 68000 transfers. Also, from the software point of view, 24-bit transfers are pseudo-transparent. From a hardware point of view, LWORD is realized in two steps:

❶ Cycle with AD01 = 0 high word

❷ Cycle with AD01 = 1 low word

Thus, for 24-bit Read and Write CAMAC cycles each type of transfer is carried out in a different way:

24-bit Read:	T=0	Branch cycle with transparent read of bits <16...23>.
	T=1	Read bits <00...15> from the temporary register.
24-bit Write:	T=0	Store bits <16...23> in the temporary register.
	T=1	Branch cycle with transparent write of bits <00...15> plus bits <16...24> from the temporary register.

For COMMAND functions the CBD 8210 only accepts these under the condition T = 1.

Note On 24-bit Read the bits <24...31> are set to 0.

3.6 GL Cycles

Two types of GL cycles exist in the CBD 8210:

❶ Automatic scan for interrupt level 3.

❷ Programmed cycle.

Automatic scan cycles are discussed in the section on interrupts.

Programmed cycles are similar to a CAMAC read. For this cycle, the crate addresses are those crates given by the BTB register.

e.g. MOVE.W(GL), (EA)

The 24-bit transfer (LWORD) is treated in the same way as a 24-bit CAMAC Read

e.g. MOVE.L(GL), (EA)

4. INTERRUPTS

4.1 General Information

The CBD 8210 contains an interrupt structure at several levels as follows:

- Two external interrupts on levels 2 and 4.
- One internal interrupt for LAM handling on level 3.

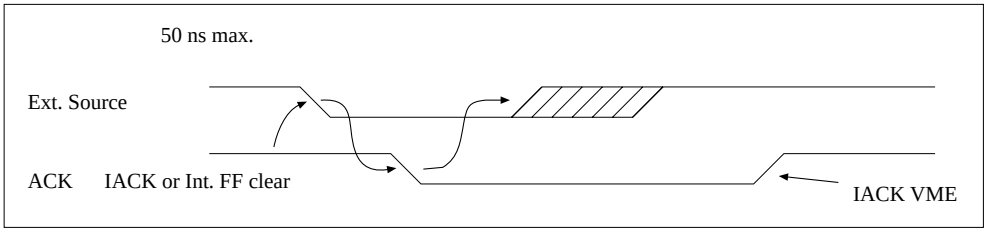
Each source of interrupt can be masked separately. The interrupt vector for levels 2 and 4 can be selected by jumpers and for level 3 by software. The interrupt logic corresponds to the VME standard revision B with a transit-time of the daisy chain of 40 ns.

Initialization sets the interrupt masks to "1" and the interrupt flags are cleared.

4.2 External Interrupts - IT2 and IT4

External interrupts (activated on the negative edge) can be selected for action by either NIM or TTL I/Ps. In addition, there is an ACK O/P allowing a handshake action with the source.

The timing diagram is as follows:



This method of operation enables external synchronization with the whole interrupt process.

The interrupt flip flop can be SET, RESET and Read by the master, allowing the use of individual masks, as follows:

❶ To use the external interrupt inputs for polling.

❷ To use the ACK output for external synchronization.

❸ To generate interrupts by software.

The vector associated with each external interrupt can be selected by jumper (8 bits) - see the section on jumper settings for vector and NIM/TTL selection.

The use of the Input / Output sockets INT and ACK allows the connection of a second CBD 8210 on the same branch with a system of handshaking.

4.3 LAM Handling

LAM handling is carried out by three interrupt controllers of type AMD 9519. This choice allows sophisticated GLAM handling with a minimum of hardware. The AMD 9519 is a complicated device, and it is recommended to study carefully the manufacturers data sheet.

The 24 GLAMs are connected to the three AMD 9519s, as follows:

GL1 to GL8	INT. CONTR. 1
GL9 to GL16	INT. CONTR. 2
GL17 to GL24	INT. CONTR. 3
(GL1 has the highest priority)	

For correct GLAM handling it is important to use the interrupt controllers, as follows:

Number of BYTE	1
Interrupt input	negative edge
GINT polarity	negative

4.3.1 GL Scanning

In order to simplify the use of the module the generation of GL cycles is done autonomously and is interleaved with CAMAC cycles. The repetition rate of GL cycles has been set to 10 µs.

The generation of BG cycles is done only if the mask MLAM = 0 and the signal BD is active.

4.3.2 Information on Interrupt Handling.

The three interrupt controllers are connected in series and the priority of the 24 sources of interrupts can be defined by three separate groups. However, group 1 has always a higher priority than group 2 which has a higher one than group 3. The vectors must be programmed before use. It is also possible to carry out interrupt polling by using the internal masks of the AMD 9519s.

It should be noted that in order not to block the action of a higher priority GL, the handling of IRQ3 VME should not stop the generation of BG cycles. Thus it is part of the requirements of the user program that it resets the AMD 9519 interrupt, which has just been handled, to zero.

5. DMA INTERFACE

From the beginning, the design of the CBD 8210 CES has been made with the intention of being able to connect a DMA controller having access to the CAMAC branch. Furthermore, side A of connector P2 has been made in order to allow the variables C, N, A and F to be fed into the module and to allow the status bits X, Q and ENDCYCLE to be sent to the DMA controller.

In addition, it is also possible to do in a single VME cycle (for example) a Read in a CAMAC module and Write into a VME memory module. The data are transmitted directly onto the VME bus.

6. FRONT PANEL

The front panel gives complete information on the type of cycles being executed. The panel is divided into three parts, as follows:

❶ Connection to the CAMAC branch.

❷ Input / Output and branch Z.

❸ LED indicators.

6.1 CAMAC Interface

This connector is the standard CAMAC BRANCH HIGHWAY connector manufactured by EMIHUGHES. It is fairly delicate and should be treated with care when connecting and disconnecting the branch highway cable. See appendix for signal layout.

6.2 BRANCH Selector Plus Input / Output Sockets

The branch selector switch, on the front panel, allows the CBD 8210 to be encoded onto the required section of the VME address field. The selector operates between 0 and 7.

External interrupts IT2 and IT4 are LEMO connectors type 00 with Vin Max = +12 V and Vin Min = -12V.

Interrupt ACK BS2 and BS4 are also LEMO type 00 and can be selected to be either:

NIM - 16 mA. to 0 mA.

or

TTL 0.5 V. to 2.7 V.

Branch ACK BRCTL output LEMO type 00

TTL Imax = 120 mA.

This output provides a pulse each time the CBD 8210 has access to the CAMAC branch, under program control. The duration of the pulse corresponds to the branch cycle time.

Push button BZ generates a branch initialize (BZ) with a duration of approximately 15 µs.

6.3 LEDs

The information given by the front panel LEDs is as follows:

- TB1 to TB7 Current "on line" crates. Corresponds to BTBx = 0
- BCR1 to BCR7 Status of the last CAMAC cycle.
- BN1 to BN16 Status of the last CAMAC cycle.
- BF1 to BF16 Status of the last CAMAC cycle.
- BA1 to BA8 Status of the last CAMAC cycle.
- X Status received during last CAMAC cycle.
- Q Status received during last CAMAC cycle.
- SY1 SY2 Read / Write bits in the CSR.
- TA BTA of the CAMAC command stretched to 0.3 S.
- BG BG of the command stretched as for BTA.
- BD Transparent state of the branch demand signal.
- TO Flag bit in the CSR if "Time-Out" during last CAMAC cycle.
- MTO Time-Out mask in the CSR.
- MLAM LAM mask bit in the CSR.

7. CBD 8210 CHARACTERISTICS

Cycle min. with branch = 1 M 1.5 µs.

Cycle max. with branch = 15 M 2.5 µs.

Cycle time for internal register = 500 ns.

VME CAMAC BRANCH DRIVER

D16 A24

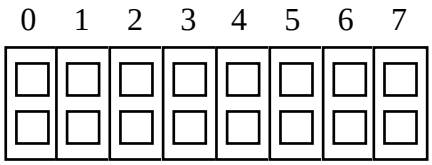
I(3), I(2), I(4)

Operating conditions	0 to 70 degrees C. 0 to 95% humidity
Supplies	+5 V. at 3.5 A (static) -12 V. at 0.1 A

Card compatible with VME rev. B.

8. JUMPER SETTINGS

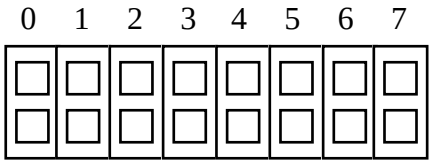
J1: Vector Number for Interrupt Level 4



IN = 1 OUT = 0

In factory set to 252

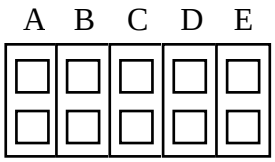
J2: Vector Number for Interrupt Level 2



IN = 1 OUT = 0

In factory set to 125

J3: Time-Out selection - variable between 2 µs and 134 s

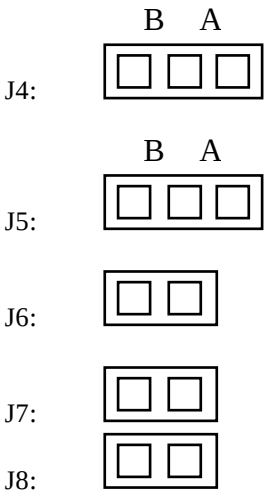


IN = L OUT = H

L H H L L = 2 µs.
H H H L L = 4 µs.
L L L H L = 8 µs.
H L L H L = 16 µs.
- - - - -
- - - - -
H H H H H = 134 s.

In factory set to 8 µs.

J4, J5, J6, J7 AND J8: Front panel level select plus ACK



Front Panel Interrupt 2 & 4

	NIM	TTL
J4	A	B
J5	A	B
J6	IN	OUT

J7	OUT	IN	ACK 2
J8	OUT	IN	ACK 4

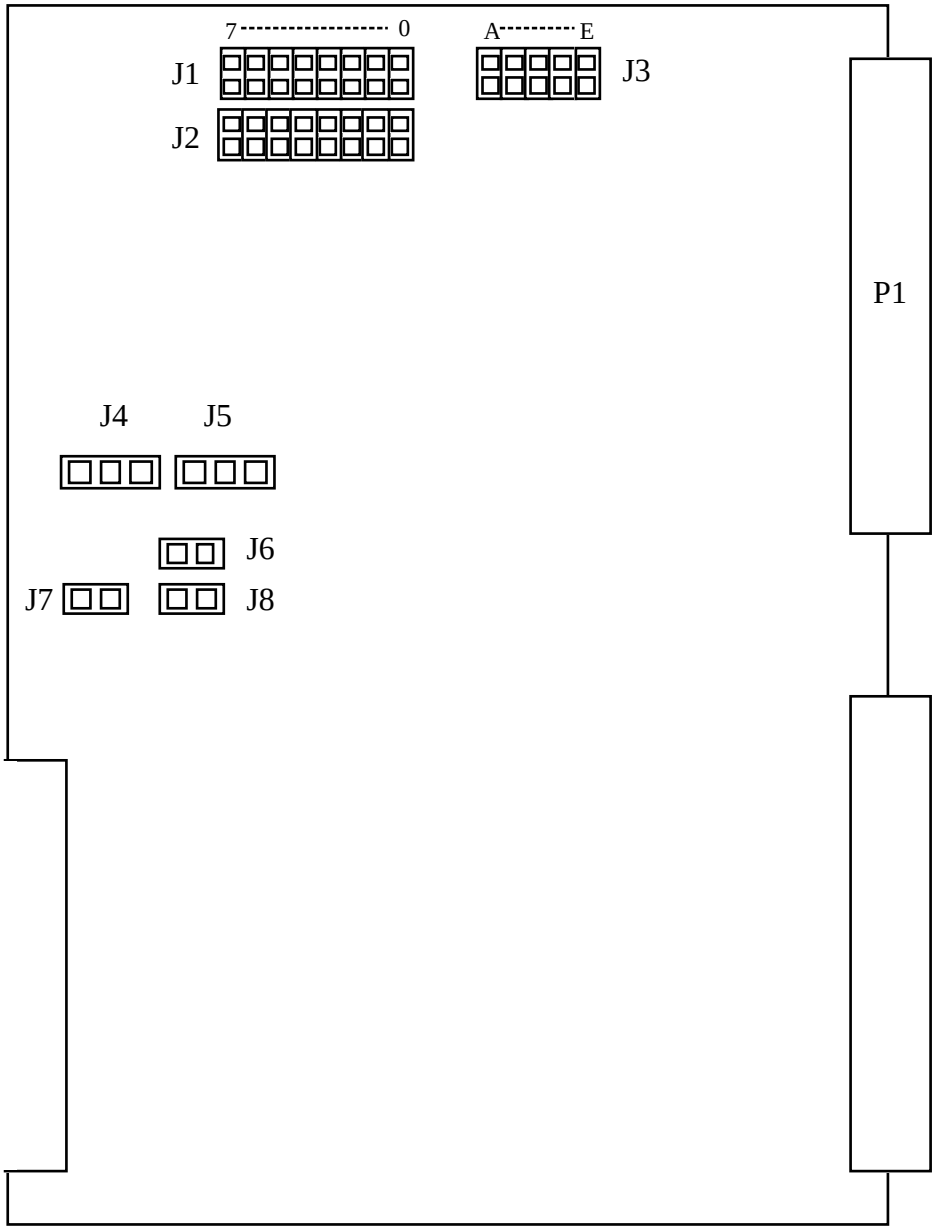
9. ANNEXES

9.1 Contact Assignments at Branch Highway Ports

Signal Contact	Return Contact	Signal	
32	13	BCR1	Crate Address
33	14	BCR2	
34	15	BCR3	
35	16	BCR4	
67	50	BCR5	
68	51	BCR6	
69	52	BCR7	
36	17	BN1	Station Address
37	18	BN2	
38	19	BN4	
39	20	BN8	
40	21	BN16	
41	1	BA1	Sub Address
23	2	BA2	
24	3	BA4	
25	4	BA8	
70	53	BF1	Function Code
71	54	BF2	
72	55	BF4	
73	56	BF8	
74	57	BF16	
61	44	BQ	Response
63	46	BTA	Timing
31	10	BTB1	
11	12	BTB2	
58	22	BTB3	
132	92	BTB4	
123	102	BTB5	
120	101	BTB6	
121	122	BTB7	
60	43	BD	Demand
59	42	BG	Graded L Request
62	45	BZ	Initialize

Signal Contact	Return Contact	Signal	
93	76	BRW1	Read / Write lines
94	77	BRW2	
95	78	BRW3	
96	79	BRW4	
97	80	BRW5	
98	81	BRW6	
99	82	BRW7	
100	83	BRW8	
103	84	BRW9	
104	85	BRW10	
105	86	BRW11	
106	87	BRW12	
107	88	BRW13	
108	89	BRW14	
109	90	BRW15	
110	91	BRW16	
112	113	BRW17	
114	115	BRW18	
116	117	BRW19	
118	119	BRW20	
124	125	BRW21	
130	131	BRW24	
26	5	BV1	Reserved lines
27	6	BV2	
28	7	BV3	
29	8	BV4	
30	9	BV5	
64	47	BV6	
65	48	BV7	
66	49	Bx	Command accepted
111	75	BSC	Cable screen

9.2 Jumpers Location



9.3 CAMAC Library

Since all of the trigger and the event builder software of the Crystal Barrel Experiment runs in VME crates and has been coded in **C-language,** it became necessary to develop a library of fast CAMAC functions usable for VME based **C-language** applications. These functions have been implemented according to the standard defined by the ESONE committee, and have been written fully in 68020 assembly language, to gain the maximum CAMAC speed (All functions have been designed to fit fully on the 68020's cache and make heavy use of the 68020's bit field instructions).

This note contains a description of CAMAC functions, callable **from C-language** using the operating system **OS-9** (Version 2.2), together with their calling sequences and the source code of their implementation (see § 9.3.1).

An example of a benchmark **C-language** program is given in § 9.3.2., which uses the 68020 CAMAC functions. This program results in 2.4 usecs per CAMAC operation in *CSUBR* block mode transfers, what corresponds to the maximum CAMAC speed possible by hardware. Due to the function calling overhead, single CAMAC operations last 15 µs.

The CAMAC Software uses the CBD 8210 branch driver from CES and has been developed on a 25 MHz Mini FORCE 21B VME system (no wait states).

Note For all Software applications, the logical parameters should be defined as:
#define **FALSE** 0
#define **TRUE** -1

In **C-language,** it is possible to pass parameters to a function in two ways: Either by reference (passing the address of an argument, like in FORTRAN) or by value (passing a copy of the argument's value).

The CCAMAC Software passes all data by reference, what is indicated by the use of the ampersand in front of corresponding variable name. Vectors of data are always passed by reference intrinsically!

CCCC (&ext)

CCCC clears the CAMAC crate defined by "*ext"*.

```
Parameters:
unsigned long    ext;
```

CCCD (&ext~&switch)

CCCD disables the CAMAC branch demand, if "*switch"* is set to FALSE; CCCD enables the crate demand, if *"switch" is* set to TRUE.

```
Parameters:
unsigned long    ext;
unsigned long    switch;
```

CCCI (&ext,&switch)

CCCI clears the CAMAC crate inhibit, if "*switch"* is set to FALSE; CCCI sets the crate inhibit, if "*switch"* is set to TRUE.

```
Parameters:
unsigned long    ext;
unsigned long    switch;
```

CCCZ (&ext)

CCCZ initializes the CAMAC crate defined by "*ext"*.

```
Parameters:
unsigned long    ext;
```

CDREG (&ext.&branch.&crate.&station.&subaddress)

CDREG combines the branch number, the crate number, the station number and the module subaddress number into the geographical address and stores the result in "*ext"*.

```
Parameters:
unsigned long    ext;
unsigned long    branch;
unsigned long    crate;
unsigned long    station;
unsigned long    subaddress;
```

CSSA (&function,&ext,&data,&q)

CSSA causes the CAMAC function specified to be executed at the CAMAC address specified by "*ext"*. This function always makes 16-bit transfers of data. The state of Q resulting from the operation is stored in "*q"*, TRUE if the operation completed successfully, or FALSE, respectively.

```
Parameters:
unsigned long    function;
unsigned long    ext;
unsigned short   data;
unsigned long    q;
```

CSUBR (&function.&ext^!data.cb)

CSUBR causes repeated CAMAC functions to be executed at the CAMAC address specified by "*ext"*. This function always makes 16-bit block transfers of data. The state of Q resulting from the operation is stored in "*cb[l]"*, TRUE if the operation completed successfully, or FALSE, respectively. The number of CAMAC operations to be executed is passed in control block "*Cb[Ol]"*.

```
Parameters:
unsigned long    function;
unsigned long    ext;
unsigned short   data[...];
unsigned long    cb[2];
```

CFSA (&function,&ext,&data,&q)

CFSA causes the CAMAC function specified to be executed at the CAMAC address specified *by "ext"*. This function always makes 24-bit transfers of data. The state of Q resulting from the operation is stored in "*q"*, TRUE if the operation completed successfully, or FALSE, respectively.

```
Parameters:
unsigned long    function;
unsigned long    ext;
unsigned long    data;
unsigned long    q;
```

CTCI (&ext.&inhibit)

CTCI tests the CAMAC crate inhibit, if "*inhibit" is* set to FALSE, the crate inhibit is off; if "*inhibit"* is returned to be TRUE, the crate is inhibited.

```
Parameters:
unsigned long    ext;
unsigned long    inhibit;
```

CTCD (&ext,&demand)

CTCD tests if a CAMAC crate demand is enabled; "*demand" is* set to TRUE, if crate demand is enabled.

```
Parameters:
unsigned long    ext;
unsigned long    demand;
```

CTGL (&ext,&demand)

CTGL tests the presence of a CAMAC crate demand; "*demand" is* set to TRUE, if crate demand is present.

```
Parameters:
unsigned long    ext;
unsigned long    demand;
```

CDLAM (&lam,&branch,&crate.,&station,inta)

CDLAM encodes all necessary values concerning a LAM. "*inta[O]"* must contain a graded LAM number, "*inta[1]"* may contain an event flag number. It combines the branch number, the crate number, the station number and stores the result in "*lam"*.

```
Parameters:
unsigned long    lam;
unsigned long    branch;
unsigned long    crate;
unsigned long    station;
unsigned long    inta[2];
```

CCLM (&lam,&switch)

CCLM enables the LAM, if "*switch" is* set to TRUE; CCLM disables the LAM, if "*switch" is* set to FALSE.

```
Parameters:
unsigned long    lam;
unsigned long    switch;
```

CTLM (&lam,&asserted)

CTLM tests the presence of a LAM, if "*asserted" is* set to FALSE, there is no LAM asserted; if "*asserted" is* returned to be TRUE, a LAM is asserted.

```
Parameters:
unsigned long    lam;
unsigned long    asserted;
```

9.3.1 CAMAC Routines

```
*****************************************************************
*
*      CAMAC routines for CES CBD 8210 Branch Driver
*      ****************************************
*      in accordance to the ESONE standard calls.
*      All parameters are long word, except data for
*      CAMAC short accesses, which are word length.
*
*      Implementation (interface to 0S9 C) of the functions
*      cccc(&ext)
*      ccci(&ext,&l)
*      cccd(&ext,&l)
*      cccz(&ext)
*      cdreg(&ext,&b,&c,&n,&a)
*      cssa(&f,&ext,&data,&q)
*      csubr(&f,&ext,data,cb)
*      cfsa(&f,&ext,&data,&q)
*      ctci(&ext,&l)
*      ctcd(&ext,&l)
*      ctgl(&ext,&l)
*      cdlam(&lam,&b,&c,&n,inta)
*      cclm(&lam,&l)
*      ctlm(&lam,&l)
*
*      Processor: 68020, CALLABLE FROM C
*      System : OS-9,Vers 2. 2
*      Programmer: M.A.Kunze,University of Karlsruhe
*      Vers .1. 2 : 28-Jul-88
*
*****************************************************************

       nam           CAMAC
       ttl           Fast 68020 CAMAC routines for 0S9 C
       psect  camac,0,0,0,0,0

*      LOGICAL*4 definition

TRUE   equ           -1

*      BITFIELD positions

BPOS   equ 19
BLEN   equ 3
BPOSB  equ 32-(BPOS+BLEN)
CPOS   equ 16
CLEN   equ 3
CPOSB  equ 32-(CPOS+CLEN)
NPOS   equ 11
NLEN   equ 5
NPOSB  equ 32-(NPOS+NLEN)
APOS   equ 7
ALEN   equ 4
APOSB  equ 32-(APOS+ALEN)
FPOS   equ 2
FLEN   equ 5
FPOSB  equ 32-(FPOS+FLEN)

*      CAMAC addresses in VME space

VMEAD  equ $FB800000 ;BASE ADDRESS FOR STANDARD VME ACCESS
SHORT  equ $00000002 ;16 BIT CAMAC ACCESS
```

```
*       STATUS REGISTER

STATUS equ VMEAD+SHORT+(29<<NPOS) ; STATUS REGISTER BRANCH 0

*       MACROS

GETPARM macro           ;GETS A PARAMETER FROM PARM. LIST
        ifeq \1-1       ;FIRST PARAMETER
        MOVEA.L D0,\2
        endc
        ifeq \1-2       ;SECOND PARAMETER
        MOVEA.L D1,\2
        endc
        ifgt \1-2       ;FURTHER PARAMETERS FROM LINKSTACK A5
        MOVEA.L (\1-1)*4(A5),\2
        endc
        endm

F           macro                           ;ASSEMBLE NAF
            MOVE.L (\1),-(A7)               ;COPY EXT TO STACK
            BFEXTU (A7){BPOSB:BLEN},D1      ;NOTE BRANCH
            MOVE.L \2,D0                    ;FUNCTION F
            BFINS D0,(A7){FPOSB:FLEN}
            MOVEA.L (A7)+,\1                ;POP EXT FROM STACK
            endm

NAF         macro                           ;ASSEMBLE NAF
            MOVE.L (\1),-(A7)               ;COPY EXT TO STACK
            BFEXTU (A7){BPOSB:BLEN},D1      ;NOTE BRANCH
            MOVE.L \2,D0                    ;STATION N
            BFINS D0,(A7){NPOSB:NLEN}
            MOVE.L \3,D0                    ;SUBADDRESS A
            BFINS D0,(A7){APOSB:ALEN}
            MOVE.L \4,D0                    ;FUNCTION F
            BFINS D0,(A7){FPOSB:FLEN}
            MOVEA.L (A7)+,\1                ;POP EXT FROM STACK
            endm

BCNAF macro                                 ;ASSEMBLE BCNAF
        MOVE.L (\1),-(A7)                   ;COPY EXT TO STACK
        MOVE.L \2,D0                        ;BRANCH
        BFINS D0,(A7){BPOSB:BLEN)
        MOVE.L \3,D0                        ;CRATE
        BFINS D0,(A7){CPOSB:CLEN}
        MOVE.L \4,D0                        ;STATION N
        BFINS D0,(A7){NPOSB:NLEN}
        MOVE.L \5,D0                        ;SUBADDRESS A
        BFINS D0,(A7){APOSB:ALEN)
        MOVE.L \6,D0                        ;FUNCTION F
        BFINS D0,(A7){FPOSB:FLEN}
        MOVE.L (A7)+,(\1)                   ;POP EXT FROM STACK
        endm
```

```
QBIT        macro                           ;CHECK THE Q-BIT
            MOVE.L #TRUE,(\1)               ;Q=TRUE
            MOVE.L #STATUS,-(A7)            ;PUSH ADDRESS OF STATUS ON STACK
            BFINS D1,(A7){BPOSB:BLEN}       ;STATUS ADDRESS FOR THIS BRANCH
            MOVEA.L (A7)+,A0
            MOVE.W (A0),D0
            BTST.L #15,D0                   ;Q RESPONSE ?
             BNE.S Q\@
            CLR.L (\1)                      ;Q=FALSE
Q\@
            endm


*************************
*       CCCC (&EXT)       *
*************************
*

CCCC: MOVEM.L A0,-(A7)                      ;SAVE REGISTERS
      GETPARM 1,A0                          ;GET ADDRESS OF EXT

      NAF A0,#28,#9,#26
      TST.W (A0)                            ;EXECUTE CAMAC OPERATION

      MOVEM.L (A7)+,A0                      ;RESTORE REGISTERS

      RTS
      align

*************************
*       CCCI (&EXT,&L) *
*************************

CCCI:  MOVEM.L A0-A1,-(A7)                  ;SAVE REGISTERS
       GETPARM 1,A0                         ;GET ADDRESS OF EXT
       GETPARM 2,A1                         ;GET ADDRESS OF L

       TST.L (A1)                           ;TEST L
       BEQ.S CCCI01                         ;FALSE=0
       NAF A0,#30,#9,#26                    ;SET INHIBIT
       BRA.S CCCI02
CCCI01 NAF A0,#30,#9,#24                    ;REMOVE INHIBIT
CCCI02 TST.W (A0)                           ;EXECUTE CAMAC OPERATION

       MOVEM.L (A7)+,A0-A1                  ;RESTORE REGISTERS

       RTS
         align
```

```
**************************
*       CCCD (&EXT,&L) *
**************************

CCCD:  MOVEM.L A0-A1,-(A7)                ;SAVE REGISTERS
       GETPARM 1,A0                       ;GET ADDRESS OF EXT
       GETPARM 2,A1                       ;GET ADDRESS OF L

       TST.L (A1)                         ;TEST L
       BEQ.S CCCD01                       ;FALSE=0
       NAF A0,#30,#10,#26                 ;ENABLE BD OUTPUT
       BRA.S CCCI02
CCCD01 NAF A0,#30,#10,#24                 ;DISABLE BD OUTPUT
CCCD02 TST.W (A0)                         ;EXECUTE CAMAC OPERATION
*
        MOVEM.L (A7)+,A0-A1 ;RESTORE REGISTERS
*
        RTS
        align

**************************
*       CCCZ (&EXT)       *
**************************

CCCZ: MOVEM.L     A0,-(A7)                ;SAVE REGISTERS
      GETPARM     1,A0                    ;GET ADDRESS OF EXT
*
      NAF         A0,#28,#8,#26           ;DATAWAY RESET
      TST.W       (A0)                    ;EXECUTE CAMAC OPERATION

      MOVEM.L     (A7)+,A0                ;RESTORE REGISTERS

      RTS
      align

************************************
*       CDREG (&EXT,&B,&C,&N,&A)      *
************************************

CDREG: LINK     A5,#00                    ;LINK TO PARAMETER LIST
       MOVEM.L  A0-A4,-(A7)               ;SAVE 5 REGISTERS
*
       GETPARM  1,A0                      ;GET ADDRESS OF EXT
       GETPARM  2,A1                      ;GET ADDRESS OF B
       GETPARM  3,A2                      ;GET ADDRESS OF C
       GETPARM  4,A3                      ;GET ADDRESS OF N
       GETPARM  5,A4                      ;GET ADDRESS OF A
*
       CLR.L    (A0)                      ;RESET EXT
       BCNAF    A0,(A1),(A2),(A3),(A4),#0
       ORI.L    #VMEAD+SHORT,(A0)         ;16 BIT ACCESS DEFAULT
*
       MOVEM.L  (A7)+,A0-A4               ;RESTORE REGISTERS
*
       UNLK     A5
       RTS
       align
```

```
*********************************
*       CSSA (&F,&EXT,&DATA,&Q)  *
*********************************

CSSA:  LINK       A5,#00                ;LINK TO PARAMETER LIST
       MOVEM.L    A0-A3,-(A7)           ;SAVE 4 REGISTERS
*
       GETPARM    1,A0                  ;GET ADDRESS OF F
       GETPARM    2,A1                  ;GET ADDRESS OF EXT
       GETPARM    3,A2                  ;GET ADDRESS OF DATA
       GETPARM    4,A3                  ;GET ADDRESS OF Q
*
       MOVE.L     (A0),D0               ;GET FUNCTION CODE
       MOVE.L     (A1),-(A7)            ;PUSH EXT ON STACK
       BFEXTU     (A7){BPOSB:BLEN},D1   ;GET BRANCH NUMBER
       BFINS      D0,(A7){FPOSB:FLEN}   ;INSERT FUNCTION
       MOVEA.L    (A7)+,A1              ;SET UP CAMAC ADDRESS
       BTST       #3,D0                 ;CONTROL?
       BNE.S      CSSA02
       BTST       #4,D0                 ;WRITE?
       BNE.S      CSSA01
       MOVE.W     (A1),(A2)             ;READ
       BRA.S      CSSA03
*
CSSA01 MOVE.W     (A2),(A1)             ;16 BIT WRITE
       BRA.S       CSSA03
*
CSSA02 TST.W      (A1)                  ;CONTROL
*
CSSA03 QBIT        A3
*
       MOVEM.L     (A7)+,A0-A3          ;RESTORE REGISTERS
*
       UNLK        A5
       RTS
       align

***********************************
*       CSUBR (&F,&EXT,&DATA,&CB)   *
***********************************
*
CSUBR:  LINK      A5,#00                ;LINK TO PARAMETER LIST
        MOVEM.L   A0-A3/D2,-(A7)        ;SAVE REGISTERS
*
        GETPARM   1,A0                  ;GET ADDRESS OF F
        GETPARM   2,A1                  ;GET ADDRESS OF EXT
        GETPARM   3,A2                  ;GET ADDRESS OF DATA
        GETPARM   4,A3                  ;GET ADDRESS OF CB
*
        MOVE.L    (A3),D2               ;GET LOOP COUNT FROM CB[0]
        SUBQ.L    #1,D2                 ;ADJUST LOOP COUNT
        BGE.S     CSUBROK               ;LOOP COUNT>0
        CLR.L     4(A3)                 ;CLEAR QBIT IN CB[1]
        BRA.S     CSUBR04               ;ERROR EXIT
*
```

```
CSUBROK MOVE.L    (A0),D0                 ;GET FUNCTION CODE
        MOVE.L    (A1),-(A7)              ;PUSH EXT ON STACK
        BFEXTU    (A7){BPOSB:BLEN},D1     ;GET BRANCH NUMBER
        BFINS     D0,(A7){FPOSB:FLEN}     ;INSERT FUNCTION
        MOVEA.L   (A7)+,A1                ;SET UP CAMAC ADDRESS
        BTST      #3,D0                   ;CONTROL?
         BNE.S    CSUBR02
        BTST      #4,D0                   ;WRITE?
         BNE.S    CSUBR01
CSUBR00 MOVE.W    (A1),(A2)+              ;READ
        DBRA      D2,CSUBR00
        BRA.S     CSUBR03
*

CSUBR01 MOVE.W    (A2)+,(A1)              ;16 BIT WRITE
        DBRA      D2,CSUBR01
        BRA.S     CSUBR03
*
CSUBR02 TST.W     (A1)      ;CONTROL
        DBRA      D2,CSUBR02
*
CSUBR03 ADDA.L    #4,A3     ;GET CB[1]
        QBIT A3
*
CSUBR04 MOVEM.L   (A7)+,D2/A0-A3          ;RESTORE REGISTERS
*
        UNLK      A5
        RTS
        align

*********************************
*       CFSA (&F,&EXT,&DATA,&Q)  *
*********************************
*
CFSA:   LINK      A5,#00                  ;LINK TO PARAMETER LIST
        MOVEM.L   A0-A3,-(A7)             ;SAVE 4 REGISTERS
*
        GETPARM   1,A0                    ;GET ADDRESS OF F
        GETPARM   2,A1                    ;GET ADDRESS OF EXT
        GETPARM   3,A2                    ;GET ADDRESS OF DATA
        GETPARM   4,A3                    ;GET ADDRESS OF Q

        MOVE.L    (A0),D0                GET FUNCTION CODE
        MOVE.L    (A1),-(A7)              ;PUSH EXT ON STACK
        BFEXTU    (A7){BPOSB:BLEN},D1     ;GET BRANCH NUMBER
        BFINS     D0,(A7){FPOSB:FLEN}     ;INSERT FUNCTION
        BCLR      #1,(A7)                 ;ENABLE LONG TRANSFER
        MOVEA.L   (A7)+,A1                ;SET UP CAMAC ADDRESS
        BTST      #3,D0                   ;CONTROL?
         BNE.S    CFSA02
        BTST      #4,D0                   ;WRITE?
         BNE.S    CFSA01
        MOVE.L    (A1),(A2)               ;24 BIT READ
         BRA.S    CFSA03
*
CFSA01  MOVE.L    (A2),(A1)               ;24 BIT WRITE
        BRA.S     CFSA03
*
CFSA02  ADDA.L    #2,A1                   ;16 BIT TRANSFER
        TST.W     (A1)                    ;CONTROL
*
CFSA03  QBIT      A3
```

```
*
        MOVEM.L   (A7)+,A3              ;RESTORE REGISTERS
*
        UNLK      A5
        RTS
        align

*************************
*       CTCI (&EXT,&L)    *
*************************
*
CTCI:   MOVEM.L   A0-A1,-(A7)           ;SAVE REGISTERS
        GETPARM   1,A0                  ;GET ADDRESS OF EXT
        GETPARM   2,A1                  ;GET ADDRESS OF L
        NAF       A0,#30,#9,#27
        TST.W     (A0)                  ;EXECUTE CAMAC OPERATION
*
        QBIT      A1
*
        MOVEM.L   (A7)+,A0-A1           ;RESTORE REGISTERS
        RTS
        align

*************************
*       CTCD (&EXT,&L)    *
*************************
*
CTCD:   MOVEM.L   A0-A1,-(A7)           ;SAVE REGISTERS
        GETPARM   1,A0                  ;GET ADDRESS OF EXT
        GETPARM   2,A1                  ;GET ADDRESS OF L
*
        NAF       A0,#30,#10,#27
        TST.W     (A0)                  ;EXECUTE CAMAC OPERATION
*
QBIT    A1
*
        MOVEM.L   (A7)+,A0-A1           ;RESTORE REGISTERS
*
        RTS
        align

*************************
*       CTGL (&EXT,&L)    *
*************************
*
CTGL:   MOVEM.L   A0-A1,-(A7)           ;SAVE REGISTERS
        GETPARM   1,A0                  ;GET ADDRESS OF EXT
        GETPARM   2,A1                  ;GET ADDRESS OF L
*
        BFEXTU    (A0){BPOSB:BLEN},D1   ;GET BRANCH NUMBER
        MOVE.L    #TRUE,(A1)            ;Q=TRUE
        MOVE.L    #STATUS,-(A7)         ;PUSH ADDRESS OF STATUS ON STACK
        BFINS     D1,(A7){BPOSB:BLEN}   ;STATUS ADDRESS FOR THIS BRANCH
        MOVEA.L   (A7)+,A0
        MOVE.L    (A0),D0
        BTST.L    #12,D0                ;BRANCH DEMAND
         BNE.S    CTGL01
        CLR.L     (A1)                  ;Q=FALSE
*
CTGL01  MOVEM.L   (A7)+,A0-A1           ;RESTORE REGISTERS
*
        RTS
        align
```

```
*
************************************
*       CDLAM (LAM,B,C,N,A,INTA)      *
************************************

CDLAM:  LINK     A5,#00                    ;LINK TO PARAMETER LIST
        MOVEM.L  A0-A5,-(A7)               ;SAVE 5 REGISTERS
*
        GETPARM  1,A0                      ;GET ADDRESS OF LAM
        GETPARM  2,A1                      ;GET ADDRESS OF B
        GETPARM  3,A2                      ;GET ADDRESS OF C
        GETPARM  4,A3                      ;GET ADDRESS OF N
        GETPARM  5,A4                      ;GET ADDRESS OF A
        GETPARM  6,A5                      ;GET ADDRESS OF GL
*
        CLR.L    (A0)                      ;RESET LAM
        BCNAF    A0,(A1),(A2),(A3),(A4),(A5)
        ORI.L    #VMEAD+SHORT,(A0)         ;16 BIT ACCESS DEFAULT
*
        MOVEM.L  (A7)+,A0-A5               ;RESTORE REGISTERS
*
        UNLK       A5
        RTS
        align

**************************
*       CTLM (LAM,L)      *
**************************
*
CTLM:   MOVEM.L    A0-A1,-(A7)             ;SAVE REGISTERS
*
        GETPARM    1,A0                    ;GET ADDRESS OF LAM
        GETPARM    2,A1                    ;GET ADDRESS OF L
*
        F          A0,#8                   ;FUNCTION 8
        TST.W      (A0)                    ;EXECUTE CAMAC OPERATION
*
        QBIT       A1
*
        MOVEM.L    (A7)+,A0-A1             ;RESTORE REGISTERS
*
        RTS
        align

**************************
*       CCLM (LAM,L)      *
**************************

CCLM:   MOVEM.L    A0-A1,-(A7)             ;SAVE REGISTERS
*
        GETPARM    1,A0                    ;GET ADDRESS OF LAM
        GETPARM    2,A1                    ;GET ADDRESS OF L
*
        TST.L      (A1)                    ;TEST L
        BEQ.S      CCLM01                  ;FALSE=0
        F          A0,#26                  ;FUNCTION 26
        BRA.S      CCLM02                  ;ENABLE LAM
CCLM01  F          A0,#24                  ;FUNCTION 24
CCLM02  TST.W      (A0)                    ;EXECUTE CAMAC OPERATION
*
        MOVEM.L (A7)+,A0-A1 ;RESTORE REGISTERS
```

```
*
        RTS
        align
*
        endsect
```

9.3.2 CAMAC Test Program

```
*********************************************************
*   CAMAC test program:
*   Transfers 16kwords of data between 'array' in a
*   VME crate and a LeCroy 4302 memory in a CAMAC crate.
*
*    Language   : C
*    System     : OS-9 Vers.2.2
*    Programmer : M.A.KUNZE,University of Karlsruhe
*    Vers.1.0   : 17-Jun-88
*********************************************************

#include <stdio.h>

main()

{

       struct { char dummy;               /* holds system time */
                char hour;
                char mins;
                char secs;
            } time;

       struct { unsigned short year;      /* holds system date */
                char month;
                char day;
            } date;

       struct { unsigned short tps;       /* holds timer information */
                unsigned short tics;
            } tick;

       short day;                         /* holds day of week */

       unsigned long  t1,t2;
       int            t3;
       float          dt;
       unsigned long  b=0,c=1,n=5,a=0,f=0; /* Location of CAMAC mem */
       unsigned long  ext=0,q=0,off=0,cb[2];
       int            blocks=1;
       unsigned short data;
       unsigned short *array;
       register
       unsigned short *pointer,i,j;

/*********************************************************/

_sysdate(2,&time,&date,&day,&tick); /* get and print date*/

printf("\nCamac Test on %2d-%2d-%4d at %2d:%2d:%2d\n",
date.day,date.month,date.year,time.hour,time.mins,time.sec)

cb[0] = 16384L;                    /* length of buffer in words */
array = 0xfb200000;                /* address of VME buffer */
```

```
pointer = array;                    /* fill buffer with ascending numbers */
for (i=0;i<cb[0];i++)
  *pointer++ = i;

/*   printf("\nEnter b c n a :");
     scanf("%1d %1d %1d ^%1d",&b,&c,&n,&a);
* /
     CDREG(&ext,&b,&c,&n,&a);
     CCCZ(&ext);
     CCCC(&ext);
     CCCI(&ext,&off);

     f = 17;                        /* switch memory to CAMAC */
     a = 1;
     data = 1;
     CDREG(&ext,&b,&c,&n,&a);
     printf("\nExt: %1x",ext);
     CSSA(&f,&ext,&data,&q);
     printf("\nQ-response switch CAMAC: %1x",q);

     a = 0;
     CDREG(&ext,&b,&c,&n,&a);
     printf("\nExt: %1x",ext);

     while    (blocks>0) {
     printf("\n\nEnter number of data blocks [%1d]: ",blocks);
     scanf("%1d",&blocks);
     printf("\nStart %d * %1d read and write transfers",blocks,cb[0]);
     f = 17;                 /* reset address pointer */
     data = 0;
     CSSA(&f,&ext,&data,&q);
     printf("\nQ-response reset address: %1x",q);

     _sysdate(3,&t1,&date,&day,&tick); /* get start time */
     t1 = t1*tick.tps + tick.tics;

#ifndef SNGL                        /* CAMAC block transfer */
      for (i=0;i<blocks;i++) {
       f = 16;       /* write upwards */
       CSUBR (&f,&ext,array,cb);
       f = 2; /* read downwards */
       CSUBR (&f,&ext,array,cb);
            }
#else         /* single CAMAC transfer */
      for (i=0;i<blocks;i++) {
       f = 16;       /* write upwards */
      for (j=0;j<cb[0];j++) CSSA (&f,&ext,array,&q);
               f = 2;               /* read downwards */
      for (j=0;j<cb[0];j++) CSSA (&f,&ext,array,&q);
            }
#endif
      _sysdate(3,&t2,&date,&day,&tick); /* get and print stop time */
        t2 = t2*tick.tps + tick.tics;
        t3 = tick.tps;
        dt = (t2-t1) * 1000000. / (float) (t3 * 2. * cb[0] * blocks);
      printf("\nQ-response data transfer: %1x\n",cb[1]);
      for (i=0;i<10;i++)
        printf("\nData %x",array[i]);
        printf("\n\nExecution time: %1d / %d secs.",(t2-t1),t3);
        printf("\n->%6.3f microsecs. / CAMAC operation",dt);
    }
}
```