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 SA - July 1996 - All Rights reserved

The reproduction of this material, in part or whole, is strictly prohibited. For copy information, please contact:

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

The information in this document is subject to change without notice. Creative Electronic Systems assumes no responsibility for any error that may appear in this document.
# CONTENTS

1. INTRODUCTION .................................................. 1
   1.1 Overview ................................................................................................. 1
   1.2 Address Mapping .................................................................................... 1
   1.3 Selection of Internal Registers .............................................................. 2

2. INTERNAL REGISTERS ........................................... 3
   2.1 Control & Status Register - CSR .......................................................... 3
   2.2 Interrupt Flag Register - IFR ............................................................... 4
   2.3 Interrupt Controller Registers - ICR ..................................................... 4
   2.4 Crate Address Register - CAR ............................................................ 4
   2.5 BTB Registers - BTB .............................................................................. 4
   2.6 BZ Generation ........................................................................................ 4
   2.7 GL Register ............................................................................................ 5
   2.8 Address Modifier .................................................................................... 5

3. CAMAC TRANSFERS .............................................. 7
   3.1 Types of Transfer .................................................................................... 7
   3.2 Control and Execution of CAMAC Cycles ............................................ 7
   3.3 CAMAC Status Information ................................................................. 8
   3.4 CAMAC Time-Out .................................................................................. 8
   3.5 24-bit CAMAC Transfers ....................................................................... 8
   3.6 GL Cycles ............................................................................................... 9

4. INTERRUPTS ........................................................ 11
   4.1 General Information .............................................................................. 11
   4.2 External Interrupts - IT2 and IT4 ......................................................... 11
   4.3 LAM Handling ...................................................................................... 12
       4.3.1 GL Scanning .................................................................................. 12
       4.3.2 Information on Interrupt Handling .............................................. 12
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:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;23...22&gt;</td>
<td>1...0</td>
</tr>
<tr>
<td>&lt;21...19&gt;</td>
<td>Branch address (0 to 7 = 8 branches)</td>
</tr>
<tr>
<td>&lt;18...16&gt;</td>
<td>Crate address 1 - 7 standard addressing.</td>
</tr>
<tr>
<td>&lt;15...11&gt;</td>
<td>N address - CAMAC station number</td>
</tr>
<tr>
<td>&lt;10...07&gt;</td>
<td>A address - CAMAC sub-address</td>
</tr>
<tr>
<td>&lt;06...02&gt;</td>
<td>F code - CAMAC function</td>
</tr>
<tr>
<td>&lt;01&gt;</td>
<td>CAMAC Word length</td>
</tr>
<tr>
<td></td>
<td>0 = 24-bit</td>
</tr>
<tr>
<td></td>
<td>1 = 16-bit</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSR</td>
<td>CR0 N29 A0 F0</td>
<td>Read / Write</td>
<td></td>
</tr>
<tr>
<td>IFR</td>
<td>CR0 N29 A0 F4</td>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>INT. CONT.1</td>
<td>CR0 N29 A0 F5</td>
<td>Read / Write</td>
<td>Control</td>
</tr>
<tr>
<td>INT. CONT.1</td>
<td>CR0 N29 A0 F1</td>
<td>Read / Write</td>
<td>Data</td>
</tr>
<tr>
<td>INT. CONT.2</td>
<td>CR0 N29 A0 F6</td>
<td>Read / Write</td>
<td></td>
</tr>
<tr>
<td>INT. CONT.2</td>
<td>CR0 N29 A0 F2</td>
<td>Read / Write</td>
<td></td>
</tr>
<tr>
<td>INT. CONT.3</td>
<td>CR0 N29 A0 F7</td>
<td>Read / Write</td>
<td></td>
</tr>
<tr>
<td>INT. CONT.3</td>
<td>CR0 N29 A0 F3</td>
<td>Read / Write</td>
<td></td>
</tr>
<tr>
<td>CAR</td>
<td>CR0 N29 A0 F8</td>
<td>Read / Write</td>
<td></td>
</tr>
<tr>
<td>BTB</td>
<td>CR0 N29 A0 F9</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>BZ</td>
<td>CR0 N29 A0 F9</td>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>GL</td>
<td>CR0 N29 A0 F10</td>
<td>Read</td>
<td></td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D09</th>
<th>D08</th>
<th>D07</th>
<th>D06</th>
<th>D05</th>
<th>D04</th>
<th>D03</th>
<th>D02</th>
<th>D01</th>
<th>D00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>X</td>
<td>TO</td>
<td>BD</td>
<td>MNOX</td>
<td>SY5</td>
<td>SY4</td>
<td>SY3</td>
<td>SY2</td>
<td>SY1</td>
<td>MTO</td>
<td>MLAM</td>
<td>MIT2</td>
<td>MIT4</td>
<td>IT2</td>
<td>IT4</td>
</tr>
</tbody>
</table>

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


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.

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D09</th>
<th>D08</th>
<th>D07</th>
<th>D06</th>
<th>D05</th>
<th>D04</th>
<th>D03</th>
<th>D02</th>
<th>D01</th>
<th>D00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>IT2</td>
<td>IT1</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D09</th>
<th>D08</th>
<th>D07</th>
<th>D06</th>
<th>D05</th>
<th>D04</th>
<th>D03</th>
<th>D02</th>
<th>D01</th>
<th>D00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>CR7</td>
<td>CR6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CR5</td>
<td>CR4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CR3</td>
<td>CR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CR1</td>
<td>-</td>
</tr>
</tbody>
</table>

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".

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D09</th>
<th>D08</th>
<th>D07</th>
<th>D06</th>
<th>D05</th>
<th>D04</th>
<th>D03</th>
<th>D02</th>
<th>D01</th>
<th>D00</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>BTB7</td>
<td>BTB6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BTB5</td>
<td>BTB4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BTB3</td>
<td>BTB2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BTB1</td>
<td>0</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D09</th>
<th>D08</th>
<th>D07</th>
<th>D06</th>
<th>D05</th>
<th>D04</th>
<th>D03</th>
<th>D02</th>
<th>D01</th>
<th>D00</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL16</td>
<td>GL15</td>
<td>GL14</td>
<td>GL13</td>
<td>GL12</td>
<td>GL11</td>
<td>GL10</td>
<td>GL09</td>
<td>GL08</td>
<td>GL07</td>
<td>GL06</td>
<td>GL05</td>
<td>GL04</td>
<td>GL03</td>
<td>GL02</td>
<td>GL01</td>
</tr>
</tbody>
</table>

T=1

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D09</th>
<th>D08</th>
<th>D07</th>
<th>D06</th>
<th>D05</th>
<th>D04</th>
<th>D03</th>
<th>D02</th>
<th>D01</th>
<th>D00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>GL24</td>
<td>GL23</td>
<td>GL22</td>
<td>GL21</td>
<td>GL20</td>
<td>GL19</td>
<td>GL18</td>
</tr>
</tbody>
</table>

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
Q0 = 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:

<table>
<thead>
<tr>
<th>Type of Transfer</th>
<th>F0 to F7</th>
<th>F15 to F16</th>
<th>F8 to F23 to F31</th>
<th>MOV (CAMAC), (EA) cycle VME.</th>
<th>MOV (EA), (CAMAC) cycle VME.</th>
<th>MOV (CAMAC), (EA) cycle VME</th>
<th>TST (CAMAC) cycle VME</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMAND functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ functions are executed by</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE functions are executed by</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMAND functions are executed by</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

1. Generating BCR, BN, BF, (DATA) BA
2. Generating BTA.
3. Waiting until all the crates addressed generate their BTBs.
4. Confirming that VME cycle can complete.
5. 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:

1. Cycle with AD01 = 0 high word
2. 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:

1. Automatic scan for interrupt level 3.
2. 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. \texttt{MOVE.W(GL), (EA)}

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

E.g. \texttt{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:

1. To use the external interrupt inputs for polling.
2. To use the ACK output for external synchronization.
3. 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 manufacturer’s data sheet.

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

<table>
<thead>
<tr>
<th>GL</th>
<th>INT. CONTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL1 to GL8</td>
<td>1</td>
</tr>
<tr>
<td>GL9 to GL16</td>
<td>2</td>
</tr>
<tr>
<td>GL17 to GL24</td>
<td>3</td>
</tr>
</tbody>
</table>

(GL1 has the highest priority)

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

<table>
<thead>
<tr>
<th>Number of BYTE</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt input</td>
<td>negative edge</td>
</tr>
<tr>
<td>GINT polarity</td>
<td>negative</td>
</tr>
</tbody>
</table>

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:

1. Connection to the CAMAC branch.
2. Input / Output and branch Z.
3. 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 I_{max} = 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**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

IN = 1  OUT = 0

In factory set to 252

**J2: Vector Number for Interrupt Level 2**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

IN = 1  OUT = 0

In factory set to 125

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

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.

In factory set to 8 µs.
**8. Jumper Settings**

**CBD 8210 - CAMAC Branch Driver**

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

![Diagram of jumpers J4, J5, J6, J7, J8]

- **J4:**
  - B: A
  - Diagram:

- **J5:**
  - B: A
  - Diagram:

- **J6:**
  - Diagram:

- **J7:**
  - Diagram:

- **J8:**
  - Diagram:

**Front Panel Interrupt 2 & 4**

<table>
<thead>
<tr>
<th></th>
<th>NIM</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>J4</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>J5</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>J6</td>
<td>IN</td>
<td>OUT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>OUT</th>
<th>IN</th>
<th>ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>J7</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>J8</td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
## 9. Annexes

### 9.1 Contact Assignments at Branch Highway Ports

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

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

**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[1]", TRUE if the operation completed successfully, or FALSE, respectively. The number of CAMAC operations to be executed is passed in control block "Cb[0]".

**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:

```c
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:

```c
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:

```c
unsigned long ext;
unsigned long demand;
```

**CDLM (&lam,&switch)**

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

Parameters:

```c
unsigned long lam;
unsigned long switch;
```

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

CDLAM encodes all necessary values concerning a LAM. "inta[0]" 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:

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

**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:

```c
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 OS9 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,\$b)
* 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 OS9 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
FPPOS 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
MOVE.L #TRUE, (\1) ;Q=TRUE
MOVE.L #STATUS,-(A7) ;PUSH ADDRESS OF STATUS ON STACK
BFINS D1, (A7) (BPOSB:BLLEN) ;STATUS ADDRESS FOR THIS BRANCH
MOVEA.L (A7)+, A0
MOVE.W (A0), D0
BSTOP L #15, D0
BNE.S Q\@
CLR.L (\1) ;Q=FALSE
Q\@
endm

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

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

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
          CCCCD02 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:  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
BIST     #3,D0；CONTROL?
BNE.S    CSSA02
BIST     #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
*
MOVE.L   (A7)+,A0-A3；RESTORE REGISTERS
*
UNLK    A5
RTS
align

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    CSUBR04；LOOP COUNT>0
CLR.L    4(A3)；CLEAR QBIT IN CB[1]
BRA.S    CSUBR04；ERROR EXIT

CSUBR04
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

CREATIVE ELECTRONIC SYSTEMS 31
* MOVEM.L (A7)+,A3 ;RESTORE REGISTERS
* UNLK A5
RTS
align

*********************************************
* CTCI (6EXT,6L) *
*********************************************
* 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 (6EXT,6L) *
*********************************************
* 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 (6EXT,6L) *
*********************************************
* 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
BST.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
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("Enter 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);
}