KERNFORSCHUNGSANLAGE JÜLICH GmbH
Zentrallabor für Elektronik

CAMAC DRIVER
unter
VAX-11/780
VMS 2.1 Betriebssystem

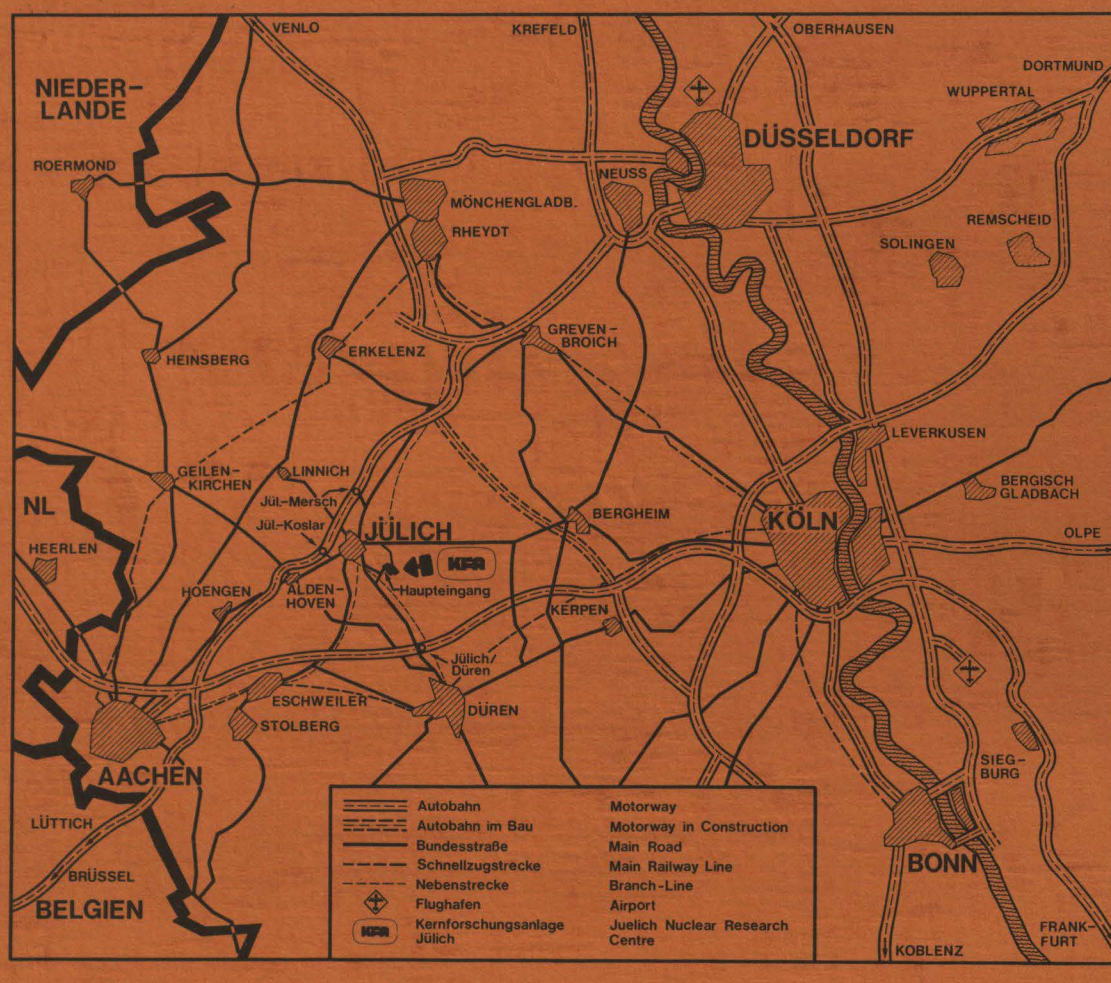
für
PDP 11 CAMAC CRATE CONTROLLER BORER 1533 A

und
DMA INTERFACE KFA - ZEL - NE 300
DISPLAY INTERFACE KFA - ZEL - NE 414

von

H. Heer, H. Stoff

Jül - Spez - 116
Juni 1981
ISSN 0343-7639



Als Manuskript gedruckt

Spezielle Berichte der Kernforschungsanlage Jülich - Nr. 116

Zentrallabor für Elektronik Jül - Spez - 116

Zu beziehen durch: ZENTRALBIBLIOTHEK der Kernforschungsanlage Jülich GmbH

Postfach 1913 · D-5170 Jülich (Bundesrepublik Deutschland)

Telefon: (02461) 61-0 · Telex: 833556 kfa d

CAMAC DRIVER

unter

VAX-11/780

VMS 2.1 Betriebssystem

für

PDP 11 CAMAC CRATE CONTROLLER BORER 1533 A

und

DMA INTERFACE KFA - ZEL - NE 300
DISPLAY INTERFACE KFA - ZEL - NE 414

von

H. Heer, H. Stoff

Inhaltsverzeichnis
****

		- Seite -
1	Einleitung	1
1.1	VAX 11 / 780 Allgemein	2
1.2	VAX 11 / 780 Hardware Konfiguration	3
2	Experiment Hardware	4
2.1	Crate-Controller	5
2.2	DMA - Interface	6
2.3	Display - Interface	7
2.4	Display - Controller	7
3	Software	8
3.1	FORTRAN Aufrufe	9
3.2	QIO - Aufrufe an den Driver	15
4	Der C A M A C Driver	16
4.1	Anweisungen an den ASSEMBLER	16
4.2	Anweisungen an den LINKER	16
4.3	Laden des Drivers	17
4.4	QIO - Funktions-Codes	19
4.5	QIO - Funktions-Modifikatoren	19
4.6	CAMAC Konstante	20
4.7	CSR und DMA Definitionen	20

- Seite -

5 Driver Codierung 21

5.1.1 Fuktionstabelle 21

5.1.2 Initialisierung I 22

5.1.3 Initialisierung II 22

5.2 F D T Routinen 24

5.2.1 DMA 24

5.2.2 DISPLAY 26

5.2.3 Lies 24 bit von CAMAC 26

5.2.4 Schreibe 24 bit nach CAMAC 27

5.2.5 Lies Crate Controller Register 28

5.2.6 Schreibe Crate Controller Register 29

5.2.7 CAMAC Funktionen 29

5.2.8 Setze UNSOLICITED L A M AST 30

5.2.9 Testhilfe - Einsprung (XDT) 30

5.3 Parameter - Test - Routine 31

5.4 IO - Routinen 32

5.4.1 DMA 33

5.4.3 Read Data & Read Controller 35

5.4.4 Write Data & Write Controller 35

5.4.5 IO Completion 36

5.4.6 Time Out 36

5.5 Interrupt Behandlung 37

5.5.1 Expected Interrupt 38

5.5.2 Unsolicited expected Interrupt 38

5.5.3 Unsolicited unexpected Interrupt 39

- Seite -

5.6 Post IO Routinen 39

5.6.1 Cancel IO 39

5.6.2 Register Dump 41

6 Test Beispiele 42

6.1 Allgemeines Testprogramm 42

6.2 DMA Testprogramm 47

6.3 Display Testprogramm 50

7 Datenraten 51

7.1 24 bit read / write Zyklus 51

7.2 DMA - Datenraten 51

8 Ausbaumoeglichkeiten 52

8.1 DMA Doppelpufferbetrieb 52

8.2 DMA Inkrementbetrieb 52

1 Einleitung

Datenerfassung mit der VAX-11/780 war eine Forderung eines Forschungsvorhabens. Datenerfassung bei der sowohl Datenmenge wie auch Datenraten Anforderungen an den Experimentrechner stellten, die sich mit den traditionellen 16 bit Minirechnern nicht erfuellen liessen. Es war die Aufgabe gestellt eigens fuer dieses Experiment entwickelte Hardware-Module in CAMAC Norm ueber einen Driver in das Betriebssystem der VAX zu integrieren. ES ist ein Software-Interface entstanden, das sicherlich auch anderen Anwendern mit aehnlich gelagerten Anforderungen eine Hilfe zur Verwirklichung sein kann.

1.2 VAX-11/780 Allgemein

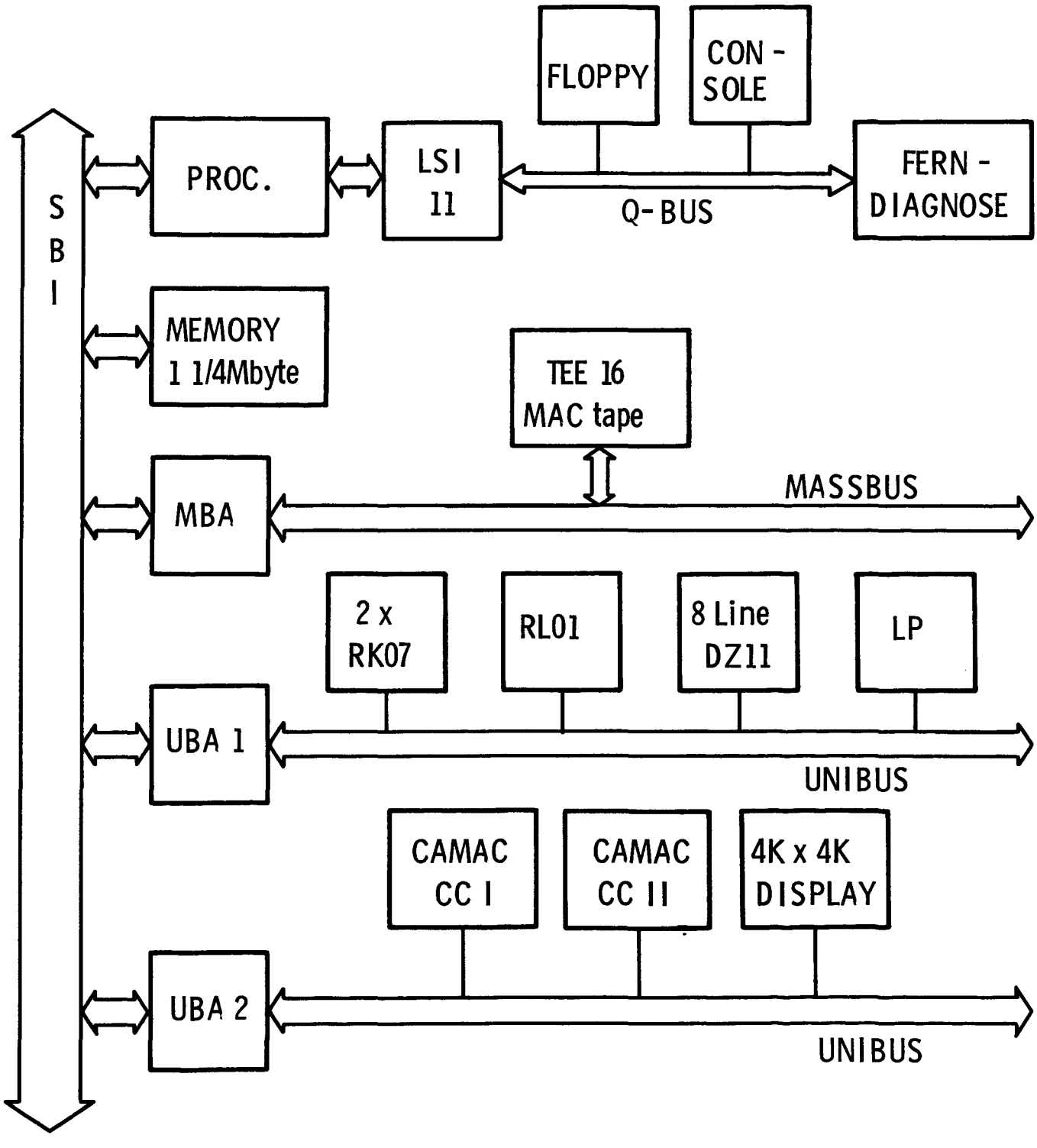
Die VAX - Familie representiert eine bedeutende Erweiterung der PDP11 Architektur. Die Bezeichnung VAX wurde von einer der wichtigsten Faehigkeiten der VAX - Familie abgeleitet, der virtuellen Adressierungsmoeglichkeit.

VAX - Virtual - ADDRESS - EXTENSION

Die Aehnlichkeit zwischen VAX11 und PDP11 Rechnern erlaubt eine einfache Uebertragung exsistierender PDP11 Software auf die VAX11. Die meisten schon bestehenden PDP11 Programme sollten unveraendert auf der VAX im PDP11 COMPATIBILITY-Mode laufen. Die VAX ist entwickelt worden, um Hochgeschwindigkeitsanwendungen zu realisieren, wobei ein fast unbegrenzter Adressraum fuer die Programme und Experimentdaten zur Verfuegung steht. Es kann ueber eine bit-Breite von 32 bit d.h. annaehernd 4 Billionen Bytes adressiert werden. Dieser Adressraum wird in einen Satz von virtuellen Adressen aufgeteilt. Die Maschinensprache der VAX baut auf dem PDP11 Assembler auf, geht aber weit ueber dessen Moeglichkeiten hinaus. Insgesammt besteht der Befehlsvorrat des VAX Assemblers aus 240 verschiedenen Basis-Befehlen. Trotdem ist dieser Assembler sehr leicht zu erlernen, da viele Befehle den Befehlen hoeherer Programmiersprachen anzugleichen sind.

1.2 VAX11-11 / 780 Hardware Konfiguration

Das unten abgebildete Blockdiagramm zeigt den Hardwareaufbau der VAX-11/780 an dem der CAMAC Driver entwickelt wurde.



2 Experiment Hardware

Fuer die verschiedenen Experimente am Juelicher Magnetspektrographen BIG KARL (1) wurde ein Vielkanalanalysatorsystem MEMPHIS (2) entwickelt.

Zur Zeit wird ein PDP15 System fuer bis zu 3 ADCs bei einer Wortlaenge von 48 bit und einer Zaehlrate von bis zu 1 x 10**4 benutzt. Verschiedene Experimente am Spektrographen haben gezeigt, dass die Anzahl der maximal moeglichen Parameter sowie die Zaehlrate nicht mehr ausreichend ist.

Die geforderte Anzahl von bis zu 16 Parametern bei einer Wortlaenge von 128 bit pro Ereignis und eine Zaehlrate von bis zu 2 x 10**4 machten es zwingend notwendig, ein modulares, jederzeit erweiterbares System aufzubauen. Spezielle Recheneinheiten fuer jeden Eingang erlauben eine Datenvorverarbeitung der ADC Informationen die den Auswerterechner die VAX11/780 entlasten.

In einem autonomen Speicher- und Displaysystem werden alle ADC Informationen parallel zum Mehrparametersystem in eigenen CAMAC Memory Modulen zu 1- oder 2-Parameterspektren inkrementiert. Das Subsystem besteht aus einem 8/16K x 24 bit CAMAC - Memory Modul pro 1 oder 2 Parameter. Diese Module koennen mit Hilfe einer neuen Datenwegoperation (4,5) von einem Controller ausgelesen und auf einem Bildschirm dargestellt werden. Ausserdem besteht die Moeglichkeit, die Daten der Memory-Module dem Auswerterechner zu uebergeben.

Referenzen
****

1) Martin, S., etal: Design Procedures for the Juelich QQDDQ High Resolution Spectrometer, Proceedings of the 5th Conference on Magnet Technology, Rome, Italy, April 21-25 (1975), 45

2) Stoff, H., Brandenburg, G., Koehler, M., Krafft, K., Mueller, K.D., Stevens, W., Teske, M.: MEMPHIS - A modular experiment multiparameter pulse height instrumentation system, IEEE Transactions on Nuclear Science Febr., NS-28, 400-404

4) ESONE Comittee, COMPEX Study Group: Compatible Extended use of the CAMAC Dataway, Draft April 21, 1980

5) Peatfield, A.C.: Extended Use of the CAMAC Dataway, IEEE Transaction on Nuclear Science, NS27 (1980), 610-611

2.1 Der CAMAC Crate Controller

Der Crate Controller Type 1533A von BORER ist ein Interface, das dazu dient, den CAMAC DATAWAY direkt an den UNIBUS eines PDP11 Computers anzuschliessen. Als doppelbreites Modul gebaut bietet der Crate Controller transparente Operationen im READ/WRITE Mode an, wobei jede Crate Subadresse wie eine Memory - Adresse erscheint. Ein weiterer Vorteil des Controllers ist die Implementierung eines Interrupt-Vektor- Generators fuer 16 Vektoren mit individueller Prioritaets Auswahl was die Interruptbearbeitung sehr schnell macht.
Zwischen dem Controller und dem Computer wird ein Handshake-Timing benutzt.

Naehere Eizelheiten zum Crate Controller sind im folgenden Manual nachzulesen :

PDP11/CAMAC CRATA CONTOLLER
TYPE 1533A
BORER
Borer Electronics AG
Solothurn 2 / Switzerland

2.2 Das DMA Interface

Das DMA CAMAC Interface ermoeglicht in Verbindung mit einem Crate-System Controller BORER 1533A den direkten Zugriff von CAMAC Modulen zum Memory einer PDP11 ohne unmittelbare Programmunterstuetzung. Dabei koennen Datenbloecke von und zum Memory im Einzel- und Doppelpufferbetrieb transferiert, angewaehlte Memoryzellen direkt inkrementiert oder beliebige Werte zu Memory-Inhalten zuaddiert werden. Alle diese Operationen beduerfen nur einer Initialisierung durch ein Programm und laufen dann autonom unter unter dem Takt der LAM-erzeugenden CAMAC Module ab. Der gleichzeitige simultane Betrieb von bis zu 4 CAMAC Modulen wird durch einen LAM-Scanner realisiert. Die Anwendung des DMA Moduls beschraenkt sich auf eine Datenwortlaenge von 16 bit bzw. 8 bit Bytes.

Naehere Einzelheiten ueber das DMA Interface siehe :

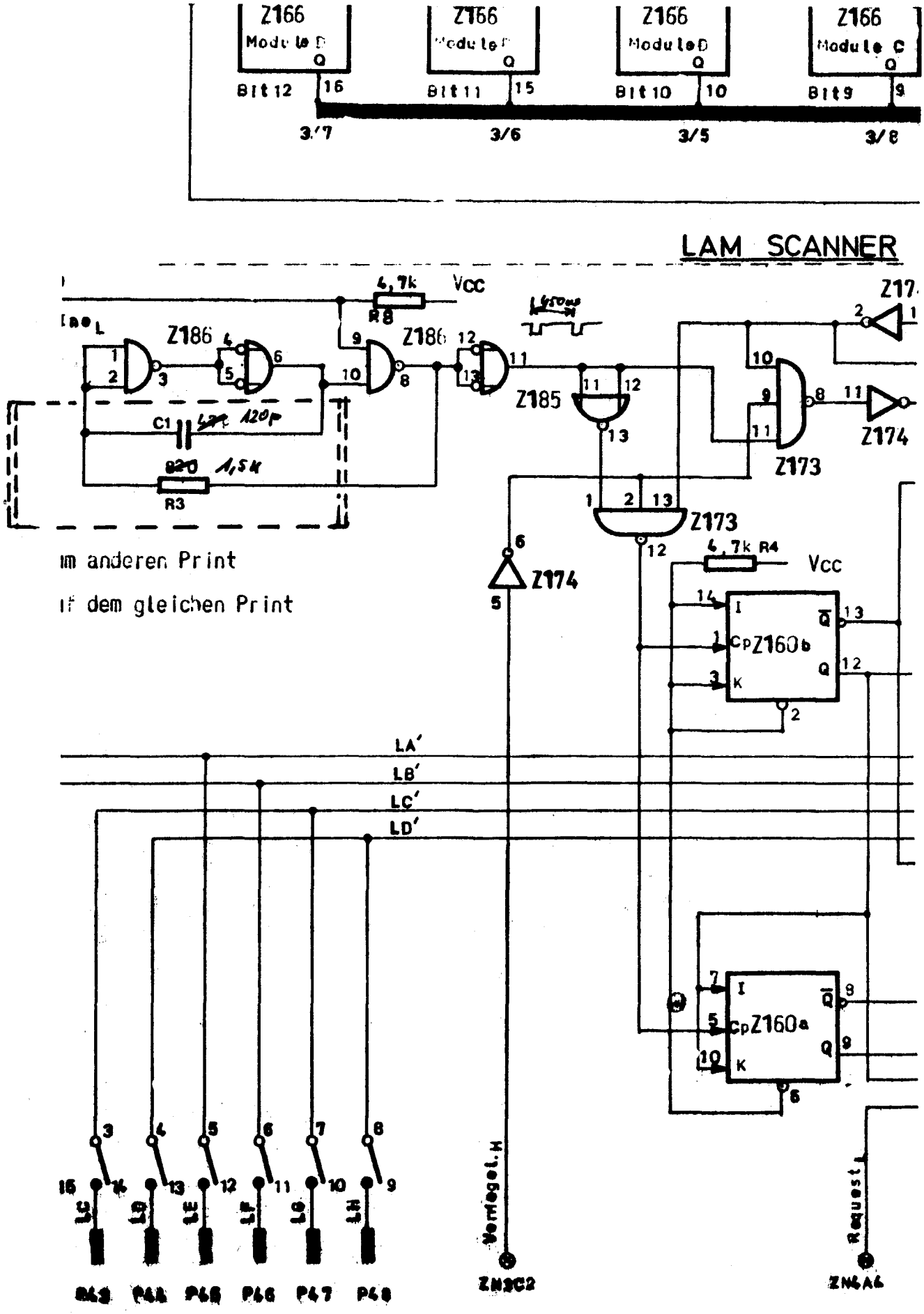
DMA - CONTROLLER

fuer CAMAC-PDP 11 System

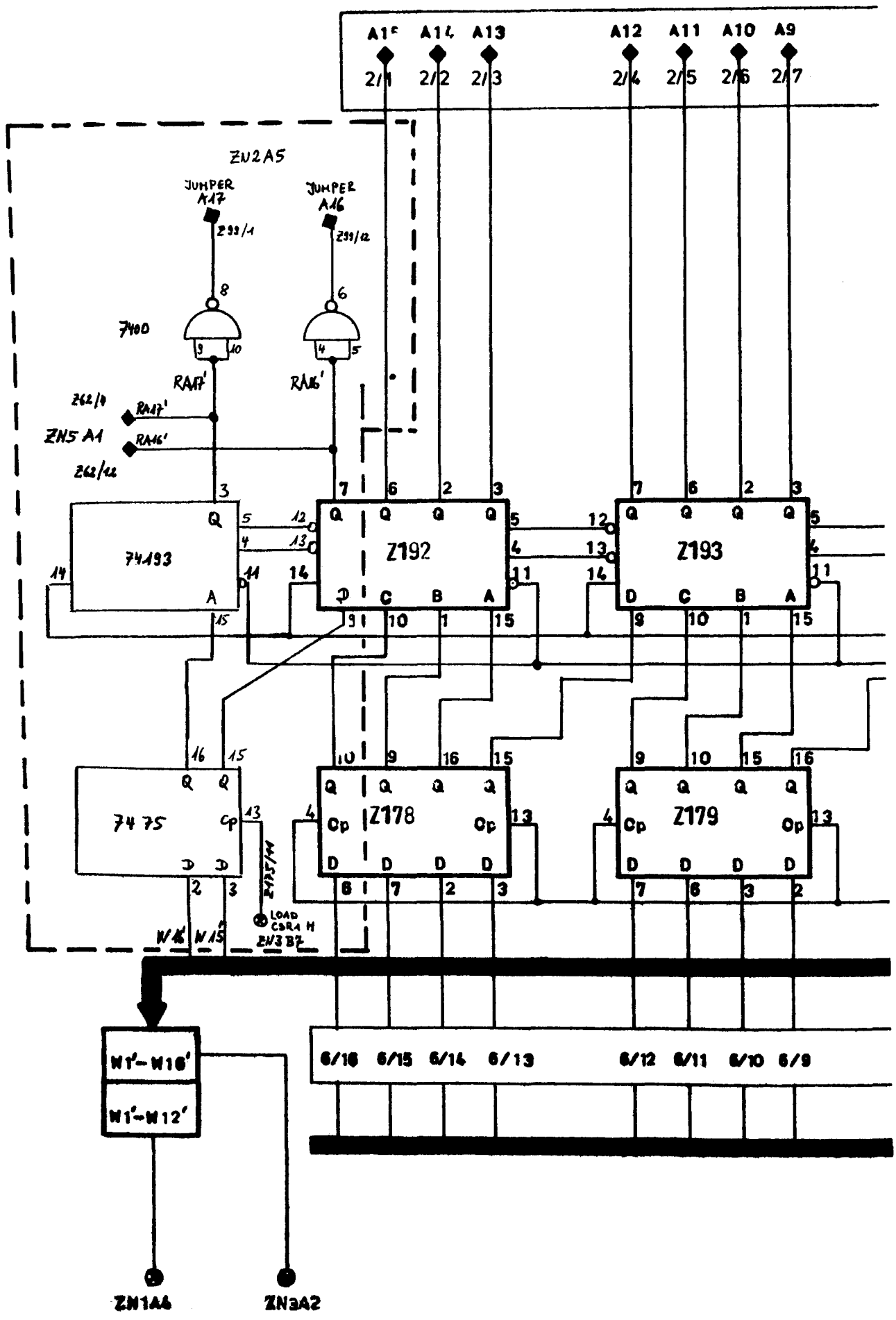
DMA-Interface

KFA-ZEL-NE-300

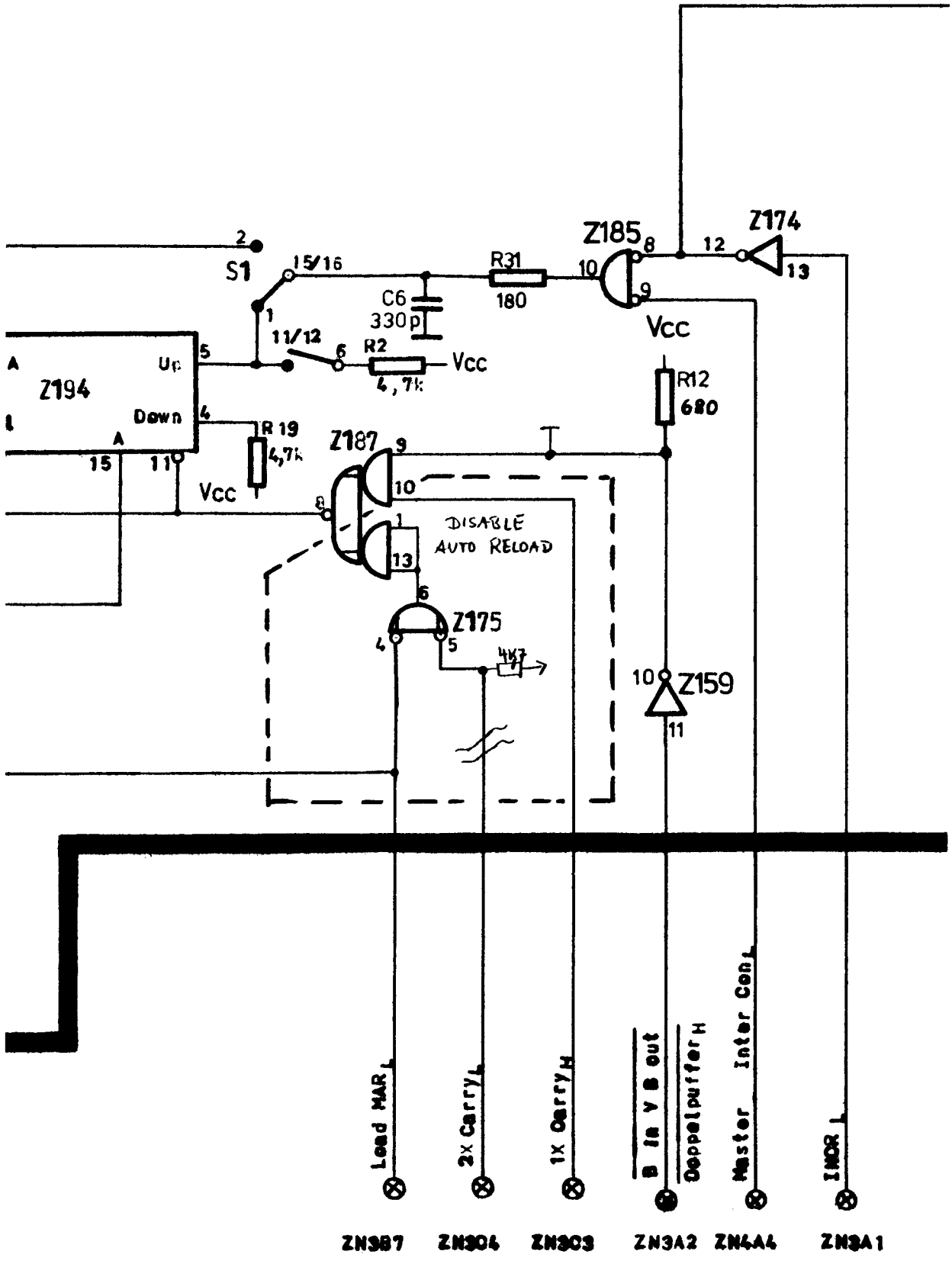
K. Zwoll, M. Chrischilles, W. John, P. Reinhart



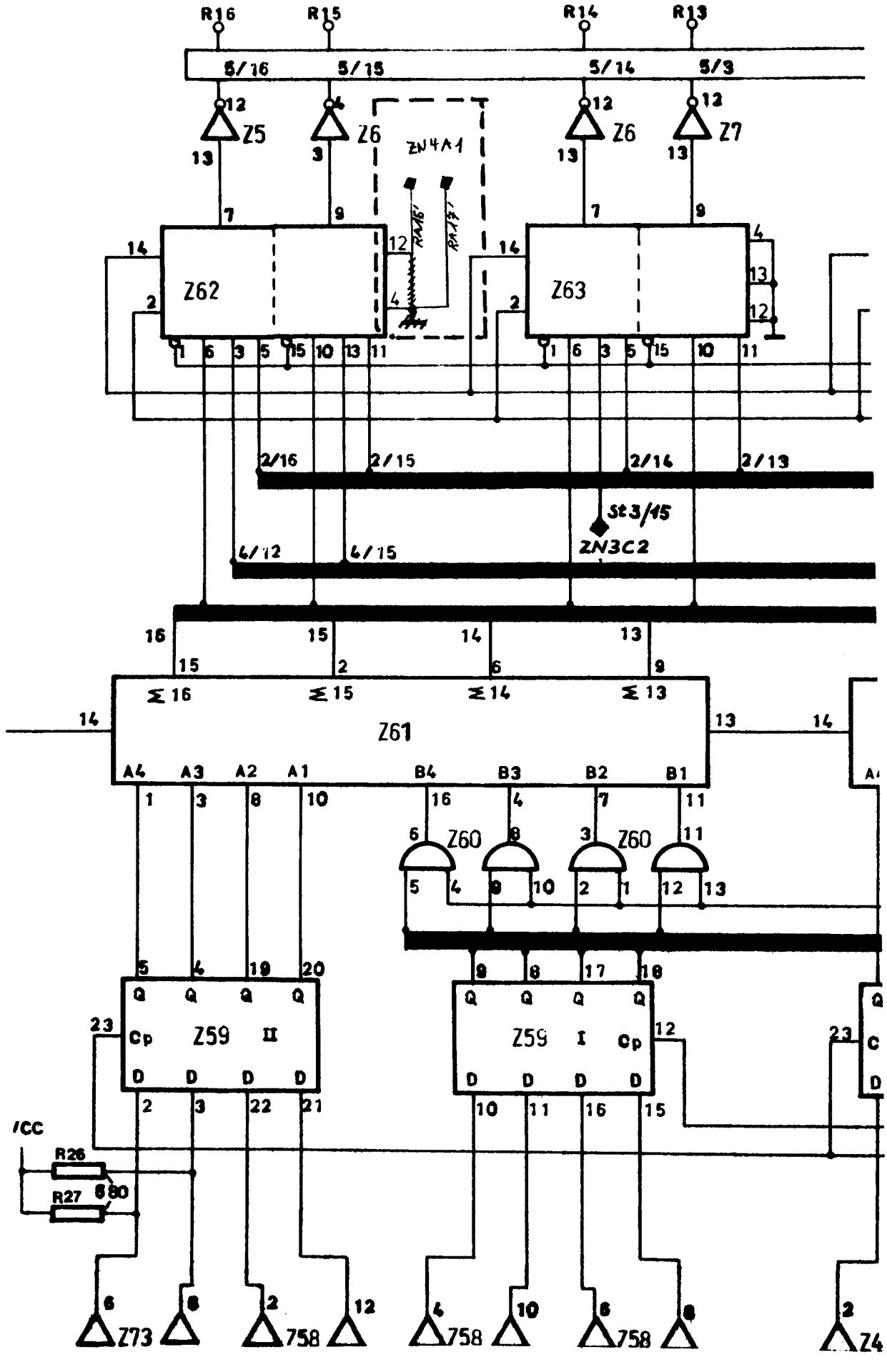
Aenderungen auf der Platine C20. Zeichnung Nr.1
DMA - Interface KFA-ZEL-NE-300



Aenderungen auf der Platine C20. Zeichnung Nr.4
DMA - Interface KFA-ZEL-NE-300



Aenderungen auf der Platine C20. Zeichnung Nr.4
DMA - Interface KFA-ZEL-NE-300



Aenderungen auf der Platine C21. Zeichnung Nr.6
DMA - Interface KFA-ZEL-NE-300

2.3 Das Display Interface

Das CAMAC Display-DMA Interface stellt die Koppeleinheit zwischen der Sichtgeraetesteuerung 'Display-Controller KFA-ZEL-NE 414' und dem CAMAC Dataway im Rahmen des vom ZEL der KFA Juelich entwickelten DISPLAY Systems her.

Weitere Einzelheiten siehe :

DISPLAY INTERFACE

KFA-ZEL-NE 414

W. John

2.4 Der DISPLAY - Controller

Der Display Controller KFA ZEL/NE 219 ist eine rein hardware-gesteuerte Sichtgeraetsteuerung ohne jede Versorgung ueber einen Programmkanal. Mechanisch ist das Geraet als 8/12 NIM-Einschub aufgebaut. Es durchlaeft sequentiell in aufsteigender Reihenfolge im Cycle-Stealing-Verfahren einen einstellbaren Adressbereich des Memories und stellt dessen jeweiligen Inhalt in Y-Richtung dar. Zur Darstellung eines Punktes bedarf es einer X und Y Information.

Einzelheiten in folgendem Manual :

DISPLAY - CONTROLLER
KFA ZEL/NE 219
R. REINARTZ
Mai/Juni 1971

3 Software

Der Driver ist im Assembler der VAX programmiert. Es wurde im Laufe der Software-Entwicklung mehr und mehr darauf geachtet, die vom Betriebssystem angebotenen Routinen einzubauen um einen wirklich systemkonformen Driver zu erhalten. Das bedeutet im Wesentlichen die Ausnutzung aller Diagnosemoeglichkeiten bei auftretenden Fehlern, die im System-Klartext dem aufrufenden Programm mitgeteilt werden. Der Driver wird ueber die sogenannten QIO - Aufrufe der einzelnen Benutzer aktiviert genau so, als wenn man ein Terminal, ein Plattenlaufwerk oder irgend ein anderes vom System unterstuetztes Geraet ansprechen will. Um den Anwendern die detaillierten Kenntnisse der QIO-Aufruf-Parameter zu ersparen, wurde ein Satz von FORTRAN Unterprogrammen erstellt, die ihrerseits den CAMAC-Driver aktivieren.

3.1 FORTRAN Aufrufe an die CAMAC - Hardware

Jeder unterschiedlichen CAMAC Aktivitaet wurde ein FORTRAN - Unterprogramm zugeordnet.

a) Zuweisung des CAMAC-Geraetekanal

b) 24 bit CAMAC lesen

c) 24 bit CAMAC schreiben

d) Datenlose CAMAC-Funktion

e) CAMAC Crate-Kontroller lesen

f) CAMAC Crate-Kontroller schreiben

g) DMA Vorbereitung und Anstoss

h) Interrupt-Verknuepfung

i) DISPLAY Vorbereitung und Anstoss

a >>> Initialisierungsvorschrift

Zuweisung des CAMAC Geraetekanals.
Will ein Programm die CAMAC Hardware ansprechen muss es durch einmaligen Aufruf des Unterprogramms *** COCA *** einen CAMAC Geraetekanal belegen. Dies muss vor dem ersten CAMAC Zugriff geschehen. (SYS$ASSIGN)

Aufruf :
******

CALL CAMAC_ASSIGN (CRMO, ICRA, IMOD, IERR)

COCA >>> COnnect to CAmac

Parameter :
****

CRMO > wird vom U.P. ermittelt und stellt die CAMAC Geraetenummer fuer ein ausgewaehltes Modul in einem angewaehlten Crate dar.

CRMO >>> CRate & MOdul

ICRA > Cratenummer (zur Zeit 1 oder 2)

IMOD > Modulnummer (1 - 23)

IERR > Status Return Block

Merke :
*****

Die CAMAC Geraetenummer CRMO muss in jedem weiteren CAMAC Aufruf an das entsprechende Modul als erster Parameter angegeben werden!

Definitionen :
****

IERR > INTEGER *2 IERR(8)

CRMO > \
ICRA > > INTEGER *2
IMOD > /

Aufbau des Status Return Blocks
****

IERR(1) = IOSB(1) > IO Completion Code

IERR(2) = IOSB(2) > Byte - Zaehler

IERR(3) = IOSB(3) > Q - CAMAC Response

IERR(4) = IOSB(4) > X - CAMAC Response

IERR(5) \
> IO Status
IERR(6) /

IERR(7) > Allgemeiner ERROR Indikator
Ø = o.k. -1 = Error

IERR(8) > vorlaeufig frei

Die einzelnen Rueckgabeparameter koennen und sollten vom aufrufenden Programm abgefragt werden. Die Anordnung der Parameter des Status Return Blocks ist bei allen CAMAC Aufrufen die gleiche.

b >> Lies CAMAC Modul (24 bit)

Aufruf : CALL CAMAC_READ (CRMO, ISUB, CARE, IDAT, IERR)
******

Parameter : CRMO > CAMAC Geraetenummer
****
ISUB > Modul - Subadresse

CARE > CAMAC READ Function

IDAT > 24 bit Daten

IERR > Status Return Block

Definitionen : CRMO \
********** ISUB > INTEGER *2
CARE /
IDAT > INTEGER *4
IERR > INTEGER *2 IERR(8)

c >> Schreibe CAMAC Modul (24 bit)

Aufruf : CALL CAMAC_WRITE (CRMO, ISUB, CAWR, IDAT, IERR)

Parameter : s.o.
****

CAWR > CAMAC WRITE Function

Definitionen : s.o.
****

CAWR > INTEGER *2

d >> Datenlose CAMAC Funktion

Aufruf : CALL CAMAC_FUNCTION (CRMO, ISUB, IFUNC, IERR)
******

Parameter : CRMO > CAMAC Geraetenummer
****
ISUB > Modul - Subadresse

IFUNC > CAMAC Funktion (Ø - 31)

IERR > Status Return Block

Definitionen : CRMO \
************ ISUB > INTEGER *2
IFUNC/
IERR > INTEGER *2 IERR(8)

e >> Lies CAMAC Contoller Status Register

Aufruf : CALL CAMAC_CSR_READ (CRMO, ICSR, IERR)
******

Parameter : CRMO \
********* > s.o.
IERR /

ICSR > CSR - Datenwort

Definitionen : CRMO \
************ > s.o.
IERR /
ICSR > INTEGER *2 ICSR(4)

f >> Schreibe CAMAC Controller Status Register

Aufruf : CALL CAMAC_CSR_WRITE (CRMO, ICSR, IERR)
******

Parameter : CRMO \
********* ICSR > s.o.
IERR /

Definitionen : CRMO \
************ ICSR > s.o.
IERR /

g >> DMA Vorbereitung und Anstoss

```
Aufruf : CALL CAMAC_DMA_READ  (CRMO,DABU,LABU,TIMO,MCOD,IERR)
         CALL CAMAC_DMA_WRITE (CRMO,DABU,LABU,TIMO,MCOD,IERR)


Parameter : CRMO > CAMAC Geraetenummer
*********   DABU > DMA - Datenpufferadresse
            LABU > Laenge des DMA - Datenpuffers
            TIMO > TIMe-Out in Sekunden
            MCOD > Modul CODe (1=A, 2=B, 3=C, 4=D)
            IERR > Status Return Block


Definitionen : CRMO > INTEGER *2
************   DABU > INTEGER *2 DABU(LABU)
                 LABU > Anzahl der 16 bit Worte
                 TIMO, MCOD > INTEGER *2
                 IERR > INTEGER *2 IERR(8)

Achtung : Bevor der DMA - Aufruf gegeben wird, muss die CAMAC
*******   Hardware entsprechend der DMA Vorstellungen konfi-
          guriert werden.  Siehe Kapitel 2.2 !
Das DMA-Interface muss aus Station 22-23 und die User-Module
muessen auf einem der 4 vorgesehenen Plaetze stecken.


Achtung : Der IOSB ist hier wie folgt aufgebaut
*******         IOSB(2) > Transfer Count
                IOSB(3) > DMA CSR1
                IOSB(4) > DMA CSR2
```

h >> CAMAC Interrupt - Verknuepfung

Erwartet man von einem CAMAC Modul einen unsolicited Interrupt (d.h. einen Interrupt zu unvorhersagbaren Zeitpunkten), so muss das entsprechende Modul mit Hilfe der UNSOLC Routine mit einer Interrupt Service Routine (ISR) verbunden werden.

Achtung : Diese Verbindung muss nach jedem aufgetretenem
*******   Interrupt von diesem Modul erneurt werden.

```
Aufruf : CALL CAMAC_INTERRUPT (CRMO, ISRU, IERR)
******

Parameter : CRMO > \
*********          > s.o.
            IERR > /

            ISRU > Adresse des Interrupt Service Unterprogr.


Definitionen : CRMO, IERR > s.o.
************
               ISRU > EXTERNAL ISRU
```

i >> Vorbereitung und Anstoss DISPLAY - Memory

Ein weiterer DMA Betrieb kann ueber den Aufruf DISPLAY angestossen werden. Hierbei werden Daten aus dem VAX-Memory im DMA-Mode auf einem Bildschirm dargestellt.

Hardwarevorraussetzung :
****

Speziell fuer unser Driverkonzept muss das Display-Interface KFA-ZEL-NE 414 im CAMAC Crate auf Station 2Ø-21 stecken. Einzelheiten zum Display und zum Display-Controller siehe Kapitel 2.3 und 2.4 !

Aufruf : CALL CAMAC_DISPLAY (CRMO, DABU, IERR)
******

Parameter : CRMO > CAMAC Geraetenummer
****
DABU > Adresse der Display-Daten

IERR > Status Return Block

Definitionen : CRMO > INTEGER *2
****
DABU > INTEGER *2 DABU(32768)

IERR > INTEGER *2 IERR(8)

Anmerkung : Alle CAMAC Unterprogramme sind im uebersetzten
****
Format in einer Bibliothek zusammengefasst. Jedes Hauptprogramm, das diese Unterprogramme aufruft, muss beim LINK-Vorgang an diese Library angebunden werden. (DNAM:[CAMAC]CAMAC.OLB)

3.2 QIO - Aufrufe an den CAMAC Driver

In den CAMAC Unterprogrammen sind die eigentlichen Aufrufe an den CAMAC Driver programmiert. Wie bei allen Driver - Aufrufen werden die QIO-Macros benutzt.
Stellvertretend fuer die verschiedenen Aufrufe in den CAMAC - Unterprogrammen soll das Beispiel aus dem Unterprogramm CAFUNC dienen.

```
STATUS = SYS$QIOW (,%VAL(CRMO)
                   ,%VAL(IO$_ACCESS)
                   ,IOSB,,,,
                   ,%VAL(IFUNC)
                   ,%VAL(ISUB),,,)
```

Parameter : CRMO > CAMAC Geraetenummer
****
IO$_ACCESS > Driver Funktionscode

IOSB > Status Return Block

IFUNC > CAMAC Funktion

ISUB > Modul - Subadresse

Merke : Hinter diesem Aufruf verbirgt sich der Assembler
*****
QIO - Aufruf der entsprechend der Parameter aus dem aufrufenden Programm versorgt werden muss.

Umseitig das komplette Listing des Unterprogramms CAFUNC.

Listing - CAMAC_FUNCTION

```
C*************************************************************************
C
C       U. P. - NAME    :       CAFUNC.FOR
C
C       BEARBEITER      :       H. HEER , H. STOFF
C
C       ERSTELLUNG      :       09-DEC-80
C
C       AENDERUNG       :       20-MAR-81
C
C       BETRIEBSSYSTEM  :       VAX11-780/VMS V2.1
C
C*************************************************************************
C
C       CAFUNC FUEHRT IM MODUL CRMO EINE CAMAC FUNCTION AUS
C
C       AUFRUF          :       CALL CAMAC_FUNCTION (CRMO, ISUB, IFUNC, IERR)
C
C       PARAMETER       :       CRMO > CAMAC CRMONUMMER
C                               ISUB  > MODUL SUBADRESSE
C                               IFUNC > CAMAC FUNCTION (0 - 31)
C                               IERR  > ALLG. STATUSBLOCK
C
        SUBROUTINE CAMAC_FUNCTION (CRMO, ISUB, IFUNC, IERR)
C
        INTEGER *2 CRMO, IFUNC, IOSB(4), IERR(8), IFLAG(2)
        INTEGER *4 STATUS, SYS$QIOW, LIB$SIGNAL
C
        EQUIVALENCE (STATUS, IFLAG(1))
C
        INCLUDE '[FORLIB]IODEF.FOR/NOLIST'
        INCLUDE '[FORLIB]SSDEF.FOR/NOLIST'
C
C       SCHREIBE CAMAC FUNKTION INS MODUL CRMO
C       **************************************
C
        STATUS = SYS$QIOW (,%VAL(CRMO), %VAL(IO$_ACCESS), IOSB,,,
        1         ,, %VAL(IFUNC), %VAL(ISUB),,,)
C
        DO 10 I=1,4
10      IERR(I) = IOSB(I)
C
        IERR(5) = IFLAG(1)
        IERR(6) = IFLAG(2)
C
        IF (.NOT. STATUS) THEN
        CALL LIB$SIGNAL (%VAL(STATUS))
        IERR(7) = -1
        RETURN
        ENDIF
C
        IF (IOSB(1) .NE. SS$_NORMAL) THEN
        CALL LIB$SIGNAL (%VAL(IOSB(1)))
        IERR(7) = -1
        RETURN
        ENDIF
C
        IERR(7) = 0
        RETURN
C
        END
```

4 Der CAMAC DRIVER

Auf den naechsten Seiten sind die wesentlichen Abschnitte des Drivers im Detail erlaeutert. Weitere Erklaerungen kann man aus dem Kommentar des Assemblerlistings entnehmen!
Grundsaetzlich besteht jeder Driver unter VMS aus einem sehr aehnlichen Skelettaufbau. Es muss nur die spezifische Hardware mit der richtigen Befehlsfolge angesprochen werden. Weiter muss die Anpassung des Unibus-Adressbereichs (18 bit) an den virtuellen Adressbereich der VAX11/780 vorgenommen werden. Fuer Anwender die in Zukunft einen CAMAC Driver entwickeln wollen, sollte der vorliegende Driver in fast allen Anspruechen ein Beispiel geben koennen.

4.1 Uebersetzung des CAMAC Drivers.

```
$MACRO/LIST CADRIVER+SYS$LIBRARY:LIB/LIBR
```

Diese Anweisung an den MACRO Compiler startet einen Uebersetzungslauf, wobei die aufgerufenen System Macros die System-Macrobibliothek SYS$LIBRARY:LIB.OLB erfordern.

4.2 Das Binden des CAMAC Drivers

```
$LINK/NOTRACE CADRIVER,CADRIVER.OPT/OPTIONS,
                        SYS$SYSTEM:SYS.STB/SELECTIVE_SEARCH
```

Mit dieser Anweisung wird der CAMAC Driver gebunden. Es werden alle Adressen zugeordnet. CADRIVER.OPT ist eine LINK-OPTION der die Transferadresse festlegt (BASE = 0).

4.3 Lade-Anweisungen fuer den CAMAC - Driver

Das Laden des CAMAC Drivers sollte nur von Personen durchgefuehrt werden, die saemmtliche Anweisungen die gegeben werden muessen bis ins Detail verstehen. (z.B. System Manager) Die Anweisungen sind unter dem System-Manager-LOGIN vorzunehmen.

1)
```
$RUN SYS$SYSTEM:SYSGEN
SYSGEN>LOAD DM1:[DRIVER]CADRIVER.EXE
```

Lade den CAMAC Driver und dessen DATA-BASE der den File-Namen CADRIVER hat von der DM1 unter dem UIC DRIVER ins System.

Hier ein paar Beispiele fuer einzelne Module im CAMAC CRATE. z.B.: Modul auf Station 7

```
SYSGEN>CONNECT CAG0/ADAP=3/VEC=%0644/CSR=%0766000/NUMVEC=1
```

Das CONNECT Kommando erstellt die DATA-BASE Controll-Blocks fuer zusaetzliche Geraete.

CAG0 >>> siehe Geraeteliste weiter unten

ADAPTER=3 ist die Nummer der SBI NEXUS - Adresse unter der der UNIBUS ADAPTER angeschlossen ist.

VEC=%0644 Diese Anweisung legt die UNIBUS Adresse des Interrupr Vektors fuer das spezielle Geraet fest. Alle Zahlen werden als Dezimalzahlen interpretiert, wenn sie nicht durch ein %O als Oktalzahl oder durch ein %X als Hexadezimalzahl deklariert werden.

CSR=%0766000 Mit der CSR Anweisung legt man die Unibus-Adresse der CAMAC Grundadresse fest.

NUMVEC=1 Hier wird die Anzahl der Interruptvektoren fuer dieses Modul festgelegt.

Die naechste Seite zeigt die CAMAC Geraetezuordnung

****

Crate #1 CSR=764000				Crate #2 CSR=766000		
MODUL "N"	DEVICE Name	VECTOR (octal)		DEVICE Name	VECTOR (octal)	
1	CAA0:	574		CBA0:	674	
2	CAB0:	570		CBB0:	670	
3	CAC0:	564		CBC0:	664	
4	CAD0:	560		CBD0:	660	
5	CAE0:	554		CBE0:	654	
6	CAF0:	550		CBF0:	650	
7	CAG0:	544		CBG0:	644	
8	CAH0:	A 1014		CBH0:	A 1054	
Reserviert fuer Test des DMA						
9	CAI0:	534		CBI0:	634	
10	CAJ0:	530		CBJ0:	630	
11	CAK0:	524		CBK0:	624	
12	CAL0:	520		CBL0:	620	
13	CAM0:	514		CBM0:	614	
14	CAN0:	510		CBN0:	610	
15	CAO0:	504		CBO0:	604	
16	CAP0:	B 1000		CBP0:	B 1040	
17	CAQ0:	C 1004		CBQ0:	C 1044	
18	CAR0:	D 500	\	CBR0:	D 600	\
19	CAS0:	540	/ Memphis IF	CBS0:	640	/ Compex IF
20	CAT0:	1020	\	CBT0:	1060	
21	-		/ Display M.	CBU0:	1064	
22	-		\	-		\
23	-		/ DMA-module	-		/ DMA-module
24	Crate controller			Crate controller		
25	Crate controller			Crate controller		

Merke: Vektor Adressen 1000 - 1074 sind 'dummy' Vektoren und werden nur benutzt, um den Driver ins SYSTEM zu integrieren. Diese Stationen koennen kein LAM an die VAX senden.

Station Nu. 8, 16, 17 und 18 koennen ein LAM an das DMA- Modul (8=A, 16=B, 17=C, 18=D) senden.

4.4 Die QIO Funktions-Codes

Der dritte Parameter bei dem QIO MACRO Aufrufen ist der QIO Funktionscode. Fuer den CAMAC Driver wurden folgende Codes zugewiesen:

```
THE QIO FUNCTIONS ARE:
**********************

IO$_READLBLK              ISSUE CAMAC CMD AND READ DATA LOW/HIGH
    WRITELBLK             WRITE DATA LOW/HIGH AND ISSUE CAMAC CMD
    ACCESS                ISSUE CAMAC CMD
    READPBLK              READ CAMAC DATA BY DMA INTO VAX MEMORY
    WRITEPBLK             WRITE DATA FROM VAX-MEMORY TO CAMAC
    SENSMODE              RETURN STANDARD PARAMETERS INTO CHAR BUFR
    SETMODE               USED TO SET UNSOLICITED LAM CONNECT
                          AND TURN ON/OFF AUTOMATIC ERROR LOGGING
    DIAGNOSE              GO TO BREAKPOINT FOR TESTING-FACILITY
```

4.5 QIO Funktions-Modifikatoren

Die einzelnen QIO Funktionen koennen durch QIO Funktions-Modifikatoren aufgeteilt werden. es werden folgende Modifikatoren benutzt.

```
THE QIO FUNCTION MODIFIERS ARE:
*******************************

IO$M_CTRLCAST             CONNECT TO LAM
IO$M_BINARY               READ OR WRITE CONTROLLER REGISTER
IO$M_REFRESH              ENABLE DISPLAY AND LOAD UBA-MAP-REGISTER
```

Die benoetigten Privileges fuer den CAMAC Benutzer sind:

PHY_IO

LOG_IO

DIAGNOSE

4.6 CAMAC Konstante

```
$DEF    CA_BASIS            .BLKW   1
$DEF    CA_INITIAL          .BLKW   1
$DEF    CA_CLEAR            .BLKW   1
$DEF    CA_MOD_SPACE1       .BLKB   698
$DEF    CA_DMA_BASIS        .BLKW   1.
$DEF    CA_DMA_CSR1         .BLKW   1.
$DEF    CA_DMA_CSR2         .BLKW   1.
$DEF    CA_DMA_MAR          .BLKW   1.
$DEF    CA_DMA_SPACE        .BLKB   4.
$DEF    CA_DMA_SET_CSR2     .BLKW   1.
$DEF    CA_DMA_RES_CSR2     .BLKW   1.
$DEF    CA_MOD_SPAADE2      .BLKB   48.
$DEF    CA_CSR              .BLKW   1
$DEF    CA_DATA_HIGH        .BLKW   1
$DEF    CA_LAM_LOW          .BLKW   1
$DEF    CA_LAM_HIGH         .BLKW   1
```

```
$DEF    UCB$W_CA_WCNT    .BLKW   1
$DEF    UCB$W_SWN        .BLKW   1
$DEF    UCB$W_CAFUNC     .BLKW   1
$DEF    UCB$W_SLOT       .BLKW   1
$DEF    UCB$W_SUBADR     .BLKW   1
$DEF    UCB$W_TIMEOUT    .BLKW   1
$DEF    UCB$B_ERRFLG     .BLKB   1
$DEF    UCB$L_ASTLHD     .BLKL   1       ;UNSOLC AST LIST HEADER
$DEF    UCB$W_CC_CSR     .BLKW   1       ;BORER CONTROLLER CSR-REGISTER
$DEF    UCB$W_DMA_CSR1   .BLKW   1       ;DMA-MODUL CSR1
$DEF    UCB$W_DMA_CSR2   .BLKW   1       ;DMA MODUL CSR2
$DEF    UCB$W_DMA_BCNT   .BLKW   1       ;DMA TRANSFERRED BYTES
$DEF    UCB$L_DMA_CA     .BLKL   1       ;DMA-MODUL START ADDRESS
$DEF    UCB$W_DMA_MODE   .BLKW   1       ;DMA-MODUL MODE_CODE
$DEF    UCB$W_DMA_LAM    .BLKW   1       ;DMA-MODUL LAM-ENABLE-CODE
```

4.7 CSR und DMA Definitionen

```
CSR DEFINITIONS AND DMA DEFINITIONS

CA$A_N1A0    = ^040
CA$M_ZINIT   = ^04
CA$M_INIMSK  = ^010
CA$M_QSTS    = ^X1
CA$M_XSTS    = ^X10000
CA$M_RINIT   = ^0300
```

5 Driver Codierung

Auf den naechsten Seiten sind die wichtigsten Abschnitte des CAMAC Drivers abgelistet. Er ist in seine Funktionsblocks aufgeteilt und jeweils mit einigen Kommentaren versehen. Dieser Weg erscheint am guenstigsten, um anderen Anwendern die Einsprungstellen zu vermitteln, an denen sie fuer ihre CAMAC Hardwarekonfiguration den Driver aendern muessen.

5.1.1 Die Funktionstabelle (FDT)

Die Driver Function Decision Tabelle treibt die geraete-abhaengige Vorverarbeitung eines I/O Aufrufes an den Driver.

```
;       FUNCTION DISPATCH TABLE
;
CA_FUNCTABLE:                   ;FUNC DECISION TABLE
;
        FUNCTAB ,-              ;ALL VALID I/O FUNCTIONS
                <READLBLK,-     ; READ DATA AND CONTROLLER
                 WRITELBLK,-    ; WRITE DATA AND CONTROLLER
                 READPBLK,-     ; DMA READ
                 WRITEPBLK,-    ; DMA WRITE
                 ACCESS,-       ; CAMAC FUNCTIONS
                 SETMODE,-      ; SET UP UNSOLICITED LAM AST
                 DIAGNOSE,-     ; TEST - FACILITY
                >
;
;
        FUNCTAB ,-              ;ALL VALID BUFFERED I/O FUNCTIONS
                <WRITELBLK,-
                 READLBLK>
;
        FUNCTAB CA_DMA_READ,-
                <READPBLK>      ; DMA READ
        FUNCTAB CA_DMA_WRITE,-
                <WRITEPBLK>     ; DMA WRITE
        FUNCTAB CA_WRITEKMAC,-
                <WRITELBLK>     ; WRITE DATA AND CONTROLLER
        FUNCTAB CA_READKMAC,-
                <READLBLK>      ; READ DATA AND CONTROLLER
        FUNCTAB CA_CAMACFUNC,-
                <ACCESS>        ; CAMAC FUNCTIONS
        FUNCTAB CA_SETMODE,-
                <SETMODE>       ; SET UP UNSOLICITED LAM AST
        FUNCTAB CA_DEBUG,-
                <DIAGNOSE>      ; GO TO BREAKPOINT FOR TEST - HELP
```

5.1.2 Initialisierung I

In dieser Initialisierungsphase wird der CAMAC Crate Controller initialisiert. In der Routine ist ein Haltepunkt implementiert, der beim Laden des Drivers angesprungen wird, wenn das VMS System mit der XDT-Testhilfe generiert wurde.

```
;
CA_CONTROL_INIT:

        JSB     G^INI$BRK

        TSTW    CA_INITIAL(R4)                  ;ISSUE Z TO CRATE
        BICW    #CA$M_INIMSK ,CA_CSR(R4)        ;CLEAR INHIBIT

        RSB
```

5.1.3 Initialisierung II

Diese Routine hat zwei Hauptfunktionen. Zum ersten testet sie ob eine gueltige CAMAC Geraetenummer gegeben wurde, und zweitens haelt sie fuer den Display - DMA Mode falls eingebaut einen direkten DATAPATH fuer die Laufzeit des Systems fest.

```
CA_UNIT_INIT:

        MOVL    UCB$L_CRB(R5), R0               ;GET POINTER TO CRB
        MOVL    CRB$L_INTD+VEC$L_IDB(R0), R1    ;GET POINTER TO IDB
        MOVL    R5, IDB$L_OWNER(R1)             ;MAKE UCB OWNER OF IDB
        CLRL    UCB$L_ASTLHD(R5)                ;CLEAR AST-LHD
;
; PICK UP DEVICE NAME, VERIFY IT HAS THREE LETTERS AND
; EXTRACT THE 1ST FOUR BITS OF THE CONTROLLER LETTER. A IS
; CONVERTED TO 0, B TO 1, C TO 2, ETC.
;
        MOVL    UCB$L_DDB(R5), R1               ;GET POINTER TO DDB
        MOVZBL  DDB$T_NAME(R1), R2              ;GET NUMBER CHARACTERS
        CMPL    #3, R2                          ;IS IT 3 ?  (EQ. CA*)
        BNEQ    10$                             ;DO NOT PUT 'ONLINE' IF NOT
        MOVL    DDB$T_NAME(R1), R1              ;GET FULL DEVICE NAME
        EXTZV   #24, #5, R1, R2                 ;GET 5 BITS (A=101, B=102..
        TSTW    R2                              ;MODUL NUMBER OUT OF RANGE
        BLEQ    10$                             ;   < 1
        CMPW    #23     ,R2                     ;   > 23
        BLSS    10$
        MOVW    R2, UCB$W_SLOT(R5)              ;SAVE SLOT NUMBER
```

```
;
2$:
;
;
        .IF DEFINED DISPLAY_INTERFACE
; IF MODUL 20 ---> SELECT HARDWARE DISPLAY
;
        CMPW    #20     ,UCB$W_SLOT(R5)
        BNEQ    3$                                      ;NOT MODUL 20

        PUSHL   R4
        MOVL    #128+2  ,R3                             ;NO. MAP-REGISTERS NEEDED
        CLRL    R4                                      ;     WE NEED 32KW


        JSB     G^IOC$ALOUBAMAPN                        ;ALLOCATE UBA MAP-REGISTERS
        POPL    R4
        BLBC    R0,     95$                             ;UNABLE TO ALLOCATE

        MOVL    UCB$L_CRB(R5)   ,R1                     ;GET CRB-ADDRESS
        TSTW    CRB$L_INTD+VEC$W_MAPREG(R1)             ;MAP REGISTER "0" ALLOCATED
        BNEQ    90$                                     ;NO

        BISB    #VEC$M_PATHLOCK, -                      ;SELECT DIRECT DATA PATH AN
                CRB$L_INTD+VEC$B_DATAPATH(R1)           ;    KEEP PATH FOREVER
        .ENDC

3$:     BISW    #UCB$M_ONLINE, UCB$W_STS(R5)            ;SET UNIT ONLINE
        BISW    #^O40   ,CA_CSR(R4)                     ;ENABLE LAM IN CCC

10$:    RSB

        .IF DEFINED DISPLAY_INTERFACE
; ERROR ON "REQUEST MAP REGISTER FOR DISPLAY"

        RELMPR                                          ;RELEASE MAP-REGISTER
95$:    MOVAB   NOLDMSG,        R1                     ;ADDRESS OF ASCIZ-MESSAGE
        PUSHR   #^M<R4,R5,R11>
        CLRL    R11                                     ;SEND MESSAGE TO CONSOLE
        JSB     G^EXE$OUTZSTRING                        ;
        POPR    #^M<R4,R5,R11>
        RSB

NOLDMSG:
        .ASCIZ  <CR><LF>/*** UNABLE TO USE 0-32KW FOR DISPLAY ***/<CR><LF>
```

5.2 Die FDT - Routinen

Jedem gueltigen QIO Funktionscode ist ein Einsprung in eine FDT Routine zugeordnet. Nachfolgend die Listen der FDT Routinen.

5.2.1 DMA

Als erstes wird das entsprechende CAMAC - DMA Modul zugeordnet.

```
CA_DMA_WRITE:
        .IF DEFINED DISPLAY_INTERFACE
        BLBS    P1(AP)  ,CA_BAD_PARAM   ;BAD BUFFER ALIGNMENT ON DDP

        BITW    #IO$M_REFRESH,  IRP$W_FUNC(R3)
        BEQL    5$                      ;NOT DISPLAY

        CMPW    #20,    UCB$W_SLOT(R5)  ;STATION 20 ?
        BNEQ    CA_BAD_PARAM            ;NO, ERROR

        JMP     G^EXE$MODIFY            ;CHECK BUFFER FOR R/W-ACCESS
5$:
        .ENDC

        MOVW    P6(AP) ,UCB$W_TIMEOUT(R5)       ;SAVE TIME OUT ?
        BNEQ    10$                             ;TIME OUT FROM USER

        MOVW    #CA_TIMEOUT_SEC, UCB$W_TIMEOUT(R5)
                                                ;SET DEFAULT SECONDS
10$:
        CMPW    #4,     P5(AP)                  ;MODUL D
        BNEQ    1$
        MOVW    #^O1000 ,R0
        MOVW    #^O600, R1
        BRB     CA_DMA_RW
1$:
        CMPW    #3,     P5(AP)                  ;MODUL C
        BNEQ    2$
        MOVW    #^O100  ,R0
        MOVW    #^O500  ,R1
        BRB     CA_DMA_RW
```

```
2$:
        CMPW    #2      ,P5(AP)                 ;MODUL B
        BNEQ    3$
        MOVW    #^010   ,R0
        MOVW    #^0440  ,R1
        BRB     CA_DMA_RW
3$:
        CMPW    #1      ,P5(AP)                 ;MODUL A
        BNEQ    CA_BAD_PARAM
        MOVW    #^01    ,R0
        MOVW    #^0420  ,R1

CA_DMA_RW:
        MOVW    R0      ,UCB$W_DMA_MODE(R5)
        MOVW    R1      ,UCB$W_DMA_LAM(R5)

        JMP     G^EXE$MODIFY


        MOVW    P6(AP) ,UCB$W_TIMEOUT(R5)       ;SAVE TIME OUT ?
        BNEQ    10$                             ;TIME OUT FROM USER

        MOVW    #CA_TIMEOUT_SEC, UCB$W_TIMEOUT(R5)
                                                ;SET DEFAULT SECONDS
10$:
        CMPW    #4,     P5(AP)                  ;MODUL D
        BNEQ    1$
        MOVW    #^02000 ,R0
        MOVW    #^0600, R1
        BRW     CA_DMA_RW
1$:
        CMPW    #3,     P5(AP)                  ;MODUL C
        BNEQ    2$
        MOVW    #^0200  ,R0
        MOVW    #^0500  ,R1
        BRW     CA_DMA_RW
2$:
        CMPW    #2      ,P5(AP)                 ;MODUL B
        BNEQ    3$
        MOVW    #^020   ,R0
        MOVW    #^0440  ,R1
        BRW     CA_DMA_RW
3$:
        CMPW    #1      ,P5(AP)                 ;MODUL A
        BNEQ    CA_BAD_PARAM
        MOVW    #^02    ,R0
        MOVW    #^0420  ,R1
        BRW     CA_DMA_RW

CA_BAD_PARAM:
```

5.2.2 DISPLAY

```
DISPLAY_MODE:

        BICB    #^077   ,CRB$L_INTD+VEC$B_DATAPATH(R3)
                                        ;SELECT DIRECT DATA PATH => NO. "0"
        LOADUBA

;  ENABLE DISPLAY

        MOVW    #26     ,UCB$W_CAFUNC(R5)       ; FUNCTION = F26
        CLRW    UCB$W_SUBADR(R5)                ; SUBADDR. = A0

        BSBW    EXE_FUNCTION                    ; ENABLE DISPLAY

        SETIPL  UCB$B_FIPL(R5)

        JMP     WAIT_COMPLETE           ;FINISH IO
        .ENDC
```

5.2.3 Lies 24 bit von CAMAC

```
CA_READKMAC:

        BITW    #IO$M_BINARY,-          ;CONTROLLER MODIFIER ?
                IRP$W_FUNC(R3)
                BNEQ    CA_RCH

        JSB     CA_PARAMCHK_READ        ;CHECK FOR LEGAL FUNC AND SUBADR
        MOVL    P1(AP)  ,R0             ;DATA LONG WORD ADDR
        MOVZBL  #CA_DATA_BUFSZ  ,R1     ;LENGTH 4 BYTES
READBUF:
        JSB     G^EXE$READCHK           ;CHECK ACCESS TO USER BUFFER
                                        ;ALSO WRITES IRP$W_BCNT
        PUSHR   #^M<R0,R3>              ;ONLY RETURNS IF CORRECT ACCESS
        ADDL2   #12     ,R1             ;INCREASE BUFR SZ FOR HEADER
        JSB     G^EXE$BUFFRQUOTA        ;CHECK USERS BUFRD IO QUOTA
        BLBC    R0      ,90$            ;BRANCH IF NO MORE QUOTA LEFT
        JSB     G^EXE$ALLOCBUF          ;TRY TO ALLOC BUFR FRM NON PAGED POO
        BLBC    R0      ,90$            ;BRANCH IF NO MORE POOL SPACE
        POPR    #^M<R0,R3>
        MOVL    R2      ,IRP$L_SVAPTE(R3) ; SAVE ADDR OF SYS BUFR
        MOVW    R1      ,IRP$W_BOFF(R3) ;AND REQUESTED BYTE COUNT
```

```
        PUSHR   #^M<R0>                 ;SAVE DATA LONG WORD ADDRESS
        MOVL    PCB$L_JIB(R4)   ,R0     ;GET JIB ADDRESS
        SUBL    R1 ,JIB$L_BYTCNT(R0)    ;ADJUST BYTE COUNT QUOTA
        POPR    #^M<R0>                 ;NOW UNSAVE IT

        MOVAB   12(R2)  ,(R2)+          ;SAVE ADDR OF START OF USER DATA
                                        ;IN FIRST LONG WORD OF BUFR HEADER
        MOVL    R0      ,(R2)           ;SAVE PROCESS BUFR ADDRS IN
                                        ;2ND LONG WORD
        JMP     G^EXE$QIODRVPKT         ;QUEUE I/O PACKET TO DRIVER
;
;       ERROR EXIT
;
90$:    POPR    #^M<R2,R3>
;
        JMP     G^EXE$ABORTIO
```

5.2.4 Schreibe 24 bit nach CAMAC

```
CA_WRITEKMAC:

        BITW    #IO$M_BINARY,-          ;CONTROLLER MODIFIER ?
                IRP$W_FUNC(R3)
                BNEQ    CA_WRITECNTRLR

        JSB     CA_PARAMCHK_WRITE       ;CHECK FOR LEGAL FUNC & SUBADR
;
        MOVL    P1(AP)  ,R0             ;DATA LONG WORD ADDR
        MOVZBL  #CA_DATA_BUFSZ ,R1      ;LENGTH (4 BYTES)
WRITEBUF:
        ASHL    #-2     ,R1     ,R9     ;NUMBER OF LONG WORDS

        JSB     G^EXE$WRITECHK          ;CHECK ACCESS TO USER BUFFER
                                        ;ALSO WRITES IRP$W_BCNT
        PUSHR   #^M<R0,R3>              ;ONLY RTNS IF CORRECT ACCESS

        ADDL2   #12     ,R1             ;INCREASE BUFR SZ FOR HEADER
```

```
        JSB     G^EXE$BUFFRQUOTA        ;CHECK USER BUFRD I/O QUOTA

        BLBC    R0      ,90$            ;BRANCH IF NOT ENOUGH QUOTA

        JSB     G^EXE$ALLOCBUF          ;TRY TO ALLOCATE BUFR FROM
                                        ;SYSTEM NON PAGED POOL
        BLBC    R0      ,90$            ;BRANCH IF NO MORE POOL SPACE
        POPR    #^M<R0,R3>
        MOVL    R2  ,IRP$L_SVAPTE(R3)   ;SAVE ADDRESS OF SYSTEM BUFR
        MOVW    R1  ,IRP$W_BOFF(R3)     ;AND BYTE COUNT

        PUSHR   #^M<R0>                 ;SAVE DATA LONG WORD ADDRESS
        MOVL    PCB$L_JIB(R4)   ,R0     ;GET JIB ADDRESS
        SUBL    R1  ,JIB$L_BYTCNT(R0)   ;ADJUST BYTE COUNT QUOTA
        POPR    #^M<R0>                 ;AND UNSAVE IT

        MOVAB   12(R2)  ,(R2)+          ;SAVE ADDR OF START OF USER DATA
                                        ;IN FIRST LONG WORD OF BUFR HEADER
        MOVL    R0      ,(R2)           ;SAVE BUFR ADDR IN 2ND LONG WORD
        ADDL2   #8.     ,R2             ;POINT TO DATA ADDR IN BUFFER
10$:    MOVL    (R0)+   ,(R2)+          ;MOV DATA TO BUFFER
        SOBGTR  R9      ,10$
        JMP     G^EXE$QIODRVPKT         ;QUEUE I/O PACKET TO DRIVER
;       ERROR EXIT
90$:    POPR    #^M<R2,R3>
        JMP     G^EXE$ABORTIO
```

5.2.5 Lies Crate Controller Register

```
;       CA_READCNTRLR
;
;       FUNCTIONAL DESCRIPTION :
;
;       READ BORER 1533 CRATE CONTROLLER REGISTERS (1 WORD/REG) AND
;       WRITE INTO BUFFER POINTED TO BY P1 (1 REG/LONG WORD) :
;       DATA LOW, DATA HIGH, LAM LOW, LAM HIGH, STATUS1, STATUS2.
;
CA_READCNTRLR:

        MOVL    P1(AP)  ,R0             ;GET BUFR ADDR
        MOVZBL  #CA_CNTRLR_BUFSZ ,R1    ;GET BUFR LENGTH

        BRW     READBUF                 ;GOTO COMMON CODE FOR ALL READS
```

5.2.6 Schreibe Crate Controller Register

```
;
;       CA_WRITECNTRLR
;
;       FUNCTIONAL DESCRIPTION :
;
;       WRITE BORER 1533 CRATE CONTROLLER REGISTERS, SAME FORMAT AS
;       FOR READCNTRLR EXCEPT IF BUFR LONG WORD IS "-1" THEN
;       DONT WRITE THAT REGISTER TO CONTROLLER.
;
CA_WRITECNTRLR:

        MOVL    P1(AP)  ,R0             ;GET BUFR ADDR
        MOVZBL  #CA_CNTRLR_BUFSZ ,R1    ;GET BUFR LENGTH

        BRB     WRITEBUF                ;GOTO COMMON CODE FOR ALL WRITES
```

5.2.7 CAMAC Funktionen

```
CA_CAMACFUNC:

        BSBW    CA_PARAMCHK_FUNCTION    ;CHECK LEGALITY OF FUNC & SUBADDR

        BSBW    EXE_FUNCTION            ;EXECUTE CAMAC FUNCTION


        CLRL    R1                      ;CLEAR X AND Q
        MOVZWL  CA_CSR(R2)      ,R0     ;CAMAC STATUS WORD INTO R0
        BITL    #^0200  ,R0             ;Q - BIT SET ?
        BEQL    20$                     ;NO >>> 20$
        BISL2   #CA$M_QSTS      ,R1     ;YES SET Q FLAG
20$:    BITL    #^0100  ,R0             ;X - BIT SET ?
        BEQL    30$                     ;NO >>> 30$
        BISL2   #CA$M_XSTS      ,R1     ;YES SET X FLAG

30$:    MOVZWL  #SS$_NORMAL, R0
        JMP     G^EXE$FINISHIO

EXE_FUNCTION:

        ASSUME  IDB$L_CSR EQ 0
        MOVL    UCB$L_CRB(R5), R0
        MOVL    @CRB$L_INTD+VEC$L_IDB(R0), R2   ; CSR -> R2

        MULW3   #CA$A_N1A0,-            ;SAVE CSR OFFSET FOR SLOT
                UCB$W_SLOT(R5), UCB$W_SWN(R5)
        MOVZWL  UCB$W_SUBADR(R5) ,R0    ;CALCULATE -
        MULW2   #2      ,R0             ;MODULE -
        ADDW2   UCB$W_SWN(R5)    ,R0    ;SUBADDR -

;       ********** CONTROL FUNCTION TO CAMAC **********
;       ***********************************************

        BICW    #7      ,CA_CSR(R2)     ;CLEAR FUNCTION BITS IN CSR
        MOVB    UCB$W_CAFUNC(R5) ,(R2)[R0] ;EXECUTE CAMAC FUNCTION

        RSB
```

5.2.8 Setze UNSOLICITED LAM AST

```
CA_SETMODE:

        BITW    #IO$M_CTRLCAST,-        ;CONNECT LAM SUBFUNCTION ?
                IRP$W_FUNC(R3)
        BEQL    80$

        CMPW    #15,    UCB$W_SLOT(R5)  ;CHECK STATION NO.
        BLSS    100$                    ;CONNECT TO LAM NOT ALLOWED !!!

        PUSHR   #^M<R4,R6,R7,R8>        ;SAVE REGS AND GET
        MOVAL UCB$L_ASTLHD(R5) ,R7      ;ADDR OF AST LIST HEAD THEN

        JSB     G^COM$SETATTNAST        ;SET UP FOR UNSOL ATTN AST

        POPR    #^M<R4,R6,R7,R8>
        BLBC    R0      ,90$
;
;       NORMAL EXIT

80$:    MOVZBL  #SS$_NORMAL ,R0

        JMP     G^EXE$FINISHIOC         ;NORMAL COMPLETION

90$:    MOVZBL  #SS$_ABORT ,R0

        JMP     G^EXE$FINISHIOC         ;ERROR COMPLETION

100$:   MOVZBL  #SS$_BADPARAM ,R0

        JMP     G^EXE$FINISHIOC         ;BAD PARAMETER COMPLETION
```

5.2.9 Testhilfe - Einsprung (XDT)

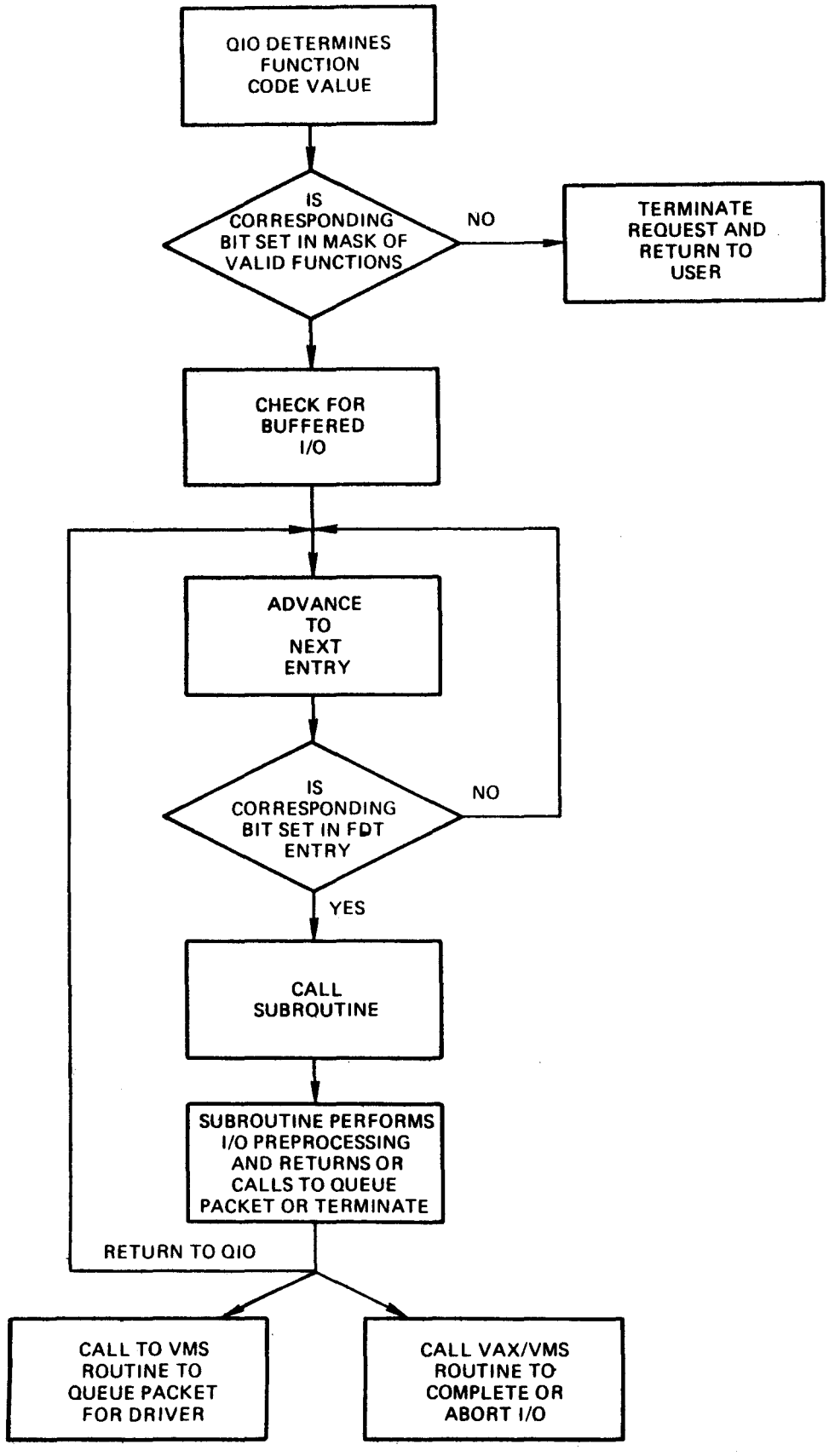
```
;       CA_DEBUG
;
;       FUNCTIONAL DESCRIPTION
;
;               ENTER XDELTA CODE
;
CA_DEBUG:

        JSB     G^INI$BRK

        MOVZBL  #SS$_NORMAL ,R0

        JMP     G^EXE$FINISHIOC
```

5.3 Parameter Test Routine

```
CA_PARAMCHK_FUNCTION:

        BITB    #8      ,P3(AP)         ;  8 <= F <= 15,  24 <= F <= 31
        BEQL    CA_BADPARAM
        BRB     CA_PARAMCHK

CA_PARAMCHK_READ:

        BITB    #24     ,P3(AP)         ;  0 <= F <= 7
        BNEQ    CA_BADPARAM
        BRB     CA_PARAMCHK

CA_PARAMCHK_WRITE:

        BITB    #16     ,P3(AP)         ;  16 <= F <= 23
        BEQL    CA_BADPARAM
        BITB    #8      ,P3(AP)
        BNEQ    CA_BADPARAM

CA_PARAMCHK:

        MOVZWL  P3(AP)  ,R0             ;GET CAMAC FUNCTION
        BLSS    CA_BADPARAM             ;ERROR, TOO SMALL
        CMPW    #CA_MAX_FUNC ,R0
        BLSS    CA_BADPARAM             ;ERROR, TOO BIG
        MOVW    R0 ,UCB$W_CAFUNC(R5)    ;SAVE CAMAC FUNCTION
        MOVZWL  P4(AP)  ,R1             ;GET SUB ADDRESS
        BLSS    CA_BADPARAM             ;ERROR, TOO SMALL
        CMPB    #CA_MAX_SUBADR ,R1
        BLSS    CA_BADPARAM             ;ERROR, TOO BIG
        MOVW    R1 ,UCB$W_SUBADR(R5)    ;SAVE SUBADDRESS

        RSB

;       ERROR RETURN
;
CA_BADPARAM:
        TSTL    (SP)+                   ;GET RID OF RTN ADDR ON STACK
        MOVZWL  #SS$_BADPARAM ,R0

        JMP     G^EXE$FINISHIOC
```

5.4 I/O Routinen

Das unten abgebildete Diagramm zeigt den Weg vom Benutzer-Programm, das mittels eines QIO - Aufrufes den Driver ueber die oben beschriebenen FDT Routinen anspringt.



5.4.1 DMA

```
DMA_WRITE_MODE:
        .IF DEFINED DISPLAY_INTERFACE
        BITW    #IO$M_REFRESH, IRP$W_FUNC(R3)
        BNEQ    DISPLAY_MODE
        .ENDC

DMA_MODE:                                       ;PREPARE FOR A BLOCK TRANSFER

        REQDPR                                  ;REQUEST A DATA PATH
        REQMPR                                  ;REQUEST A SET OF MAP REGISTERS
        LOADUBA                                 ;LOAD THE MAP REGISTERS

        CALCULATE UNIBUS START ADDRESS & THE WORD COUNT OF THE TRANSFER

        DIVW3   #2, UCB$W_BCNT(R5), UCB$W_CA_WCNT(R5)
        MNEGW   UCB$W_CA_WCNT(R5), R0
        SUBW2   #1, R0                          ;BECAUSE DMA-MODUL NEEDS IT
        MOVW    R0, CA_DMA_CSR2(R4)

        MOVZWL  UCB$W_BOFF(R5), R1              ; BYTE OFFSET IN FIRST PAGE
                                                ; OF TRANSFER
        MOVL    UCB$L_CRB(R5), R2               ; ADDRESS OF CRB
        INSV    CRB$L_INTD+VEC$W_MAPREG(R2), #9, #9, R1
                                                ; INSERT PAGE NUMBER
        SUBL2   #2, R1                          ;BECAUSE DMA-MODUL NEEDS IT
        MOVL    R1, UCB$L_DMA_CA(R5)            ;SAVE START ADDRESS
                                                ; TO CALCULATE FINAL TRANFER COUNT
        EXTZV   #16, #2, R1, R2                 ; EXTRACT BITS 17:16 OF BUS ADDRESS
        ASHL    #14, R2, R2                     ; SHIFT EXTENDED MEMORY BITS

        DSBINT

        BISW    UCB$W_DMA_MODE(R5), R2          ;ENABLE MODUL-MODE A,B,C,D
        MOVW    R2 ,CA_DMA_CSR1(R4)             ;WRITE CSR1 AND EXT. MEMORY BITS
        MOVW    R1, CA_DMA_MAR(R4)              ; WRITE CURRENT ADDRESS

        ATTENTION : THIS TRANSFERS THE ADDRESS-BITS 15 & 16 TOO
        ************************************************

        MOVW    UCB$W_DMA_LAM(R5), CA_DMA_SET_CSR2(R4)

        BICW    #7      ,CA_CSR(R4)             ;CLEAR FUNCTION BITS IN CSR
        BISW    #^040   ,CA_CSR(R4)             ;ENABLE INTERRUPT FOR CCC
                                                ; IF NOT ALREADY DONE
```

```
; WRITE THIS UCB-ADDRESS INTO IDB$L_OWNER FOR INTERRUPT ROUTINE:

        MOVL    UCB$L_CRB(R5), R0               ;GET POINTER TO CRB
        MOVL    CRB$L_INTD+VEC$L_IDB(R0), R1    ;GET POINTER TO IDB
        MOVL    R5, IDB$L_OWNER(R1)             ;MAKE UCB OWNER OF IDB


        MOVZWL  UCB$W_TIMEOUT(R5), R0   ;GET TIMEOUT IN SECONDS
        WFIKPCH CA_DEV_TIMEOUT, R0      ;WAIT FOR INTERRUPTS


;       ********** DEVICE HAS INTERRUPTED >> F O R K

        IOFORK

        PURDPR                          ;PURGE UBA BUFFERED DATA PATH
        RELMPR                          ;RELEASE MAP-REGISTERS
        RELDPR                          ;RELEASE DATA PATH

;       TEST IO-COMPLETION:

        MOVZWL  #SS$_NORMAL, R0         ;SUCCESSFUL COMPLETION CODE

        TSTW    UCB$W_DMA_CSR2(R5)      ; WC-OVERFLOW SET ???
        BLSS    10$                     ; YES, OK

        MOVZWL  #SS$_OPINCOMPL, R0      ;WORD COUNT OVERFLOW NOT SET !!!
                                        ;=> OPERATION INCOMPLETE
10$:
        INSV    UCB$W_DMA_BCNT(R5), #16, #16, R0
                                        ;FINAL TRANSFER COUNT

        MOVL    UCB$W_DMA_CSR1(R5), R1  ;RETURN  CSR1 AND  CSR2  TO USER

        BISW    #^O40   ,CA_CSR(R4)     ;ENABLE INTERRUPT FOR CCC

        REQCOM                          ;FINISH REQUEST IN EXEC
```

5.4.3 Read Data & Read Controller

```
;       ********** READ 24 BIT FROM CAMAC MODUL **********
;       ************************************************************

QIO_READ_DATA:
        MOVW    (R4)[R2]        ,(R0)    ;READ LOW 16 BITS TO DATA LOW
        MOVW    CA_DATA_HIGH(R4) ,2(R0) ;READ HIGH MANTISSA TO DATA HIGH
        BRW     NORM_COMPLETE

;       ********** READ CONTROLLER REGISTER **********
;       ************************************************

QIO_READ_CSR:
        MOVW    CA_CSR(R4)      ,(R0)+

        MOVW    CA_DATA_HIGH(R4),(R0)+
        MOVW    CA_LAM_LOW(R4)  ,(R0)+
        MOVW    CA_LAM_HIGH(R4) ,(R0)+
        BRW     NORM_COMPLETE
```

5.4.4 Write Data & Write Controller

```
;       ********** WRITE 24 BIT TO CAMAC MODUL **********
;       ************************************************************

QIO_WRITE_DATA:

        MOVW    2(R0)   ,CA_DATA_HIGH(R4) ;WRITE HIGH MANTISSA
        MOVW    (R0)    ,(R4)[R2]         ;WRITE LOW 16 BIT'S TO MODULE
        BRW     NORM_COMPLETE

;       ********** WRITE  C A M A C  CONTROLLER **********
;       ************************************************************
;
QIO_WRITE_CSR:

        MOVW    (R0)    ,R2               ;GET USER DATA
        BLSS    10$
        BICW    #CA_CSR_MASK    ,R2       ;CLEAR JUNK
        BISW    R2      ,CA_CSR(R4)
        BRB     NORM_COMPLETE
10$:
        MNEGW   R2,     R2                ;MAKE POSITIV
        BICW    #CA_CSR_MASK    ,R2       ;CLEAR JUNK
        BICW    R2      ,CA_CSR(R4)
        BRB     NORM_COMPLETE
```

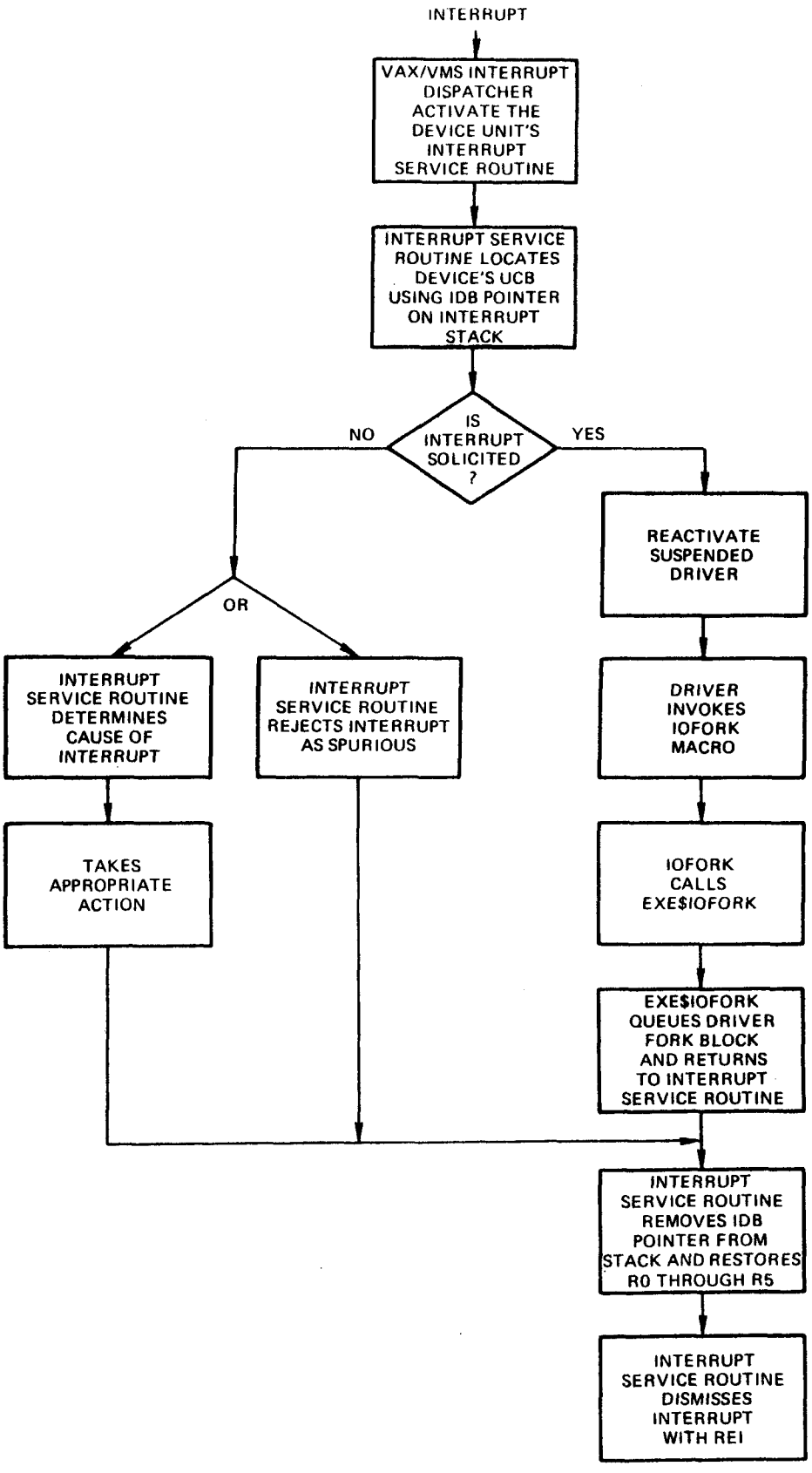
5.4.5 IO Completion

```
NORM_COMPLETE:

        ENBINT                                  ;RETURN WITHOUT LAM WAIT
;       ****************************************************************

WAIT_COMPLETE:                                  ;RETURN AFTER WFIRLCH

        CLRL    R1                              ;SET UP XQ DEV DEP STATUS
        MOVZWL  CA_CSR(R4)      ,R0             ;CAMAC STATUS WORD INTO R0
        BITL    #^0200  ,R0                     ;Q - BIT SET ?
        BEQL    20$                             ;NO >>> 20$
        BISL2   #CA$M_QSTS      ,R1             ;YES SET Q FLAG
20$:    BITL    #^0100  ,R0                     ;X - BIT SET ?
        BEQL    30$                             ;NO >>> 30$
        BISL2   #CA$M_XSTS      ,R1             ;YES SET X FLAG
30$:    MOVZWL  #SS$_NORMAL     ,R0             ;NORMAL RETURN STATUS
        INSV    UCB$W_BCNT(R5)  ,#16,-          ;# BYTES READ WRITE
                #16     ,R0

COMPLETE_IO:

        REQCOM
```

5.4.6 Time Out

```
CA_DEV_TIMEOUT:

        SETIPL  UCB$B_FIPL(R5)

        PURDPR
        RELMPR
        RELDPR

CA_DEV_TIMEOUTW:

        MOVL    UCB$L_CRB(R5)   , R4    ;CALCULATE  C S R  ADDRESS
        MOVL    @CRB$L_INTD+VEC$L_IDB(R4), R4

        BSBW    CA_REGISTER             ;SAVE SOME REGISTERS

        MOVW    #^0163777, CA_DMA_RES_CSR2(R4) ;RESET CSR2 BITS !!!!!!
        CLRW    CA_DMA_CSR1(R4)                ;CLEAR CSR1


POWER:

        MOVZWL  #SS$_TIMEOUT    ,R0

        INSV    UCB$W_DMA_BCNT(R5), #16, #16, R0
                                        ;FINAL TRANSFER COUNT

        MOVL    UCB$W_DMA_CSR1(R5), R1  ;RETURN  CSR1  AND  CSR2  TO USER

        BICW    #<UCB$M_TIM!UCB$M_INT!UCB$M_TIMOUT!UCB$M_CANCEL!UCB$M_POWER>
                UCB$W_STS(R5)           ;CLEAR UNIT STATUS FLAGS

        REQCOM
```

5.5 Interruptbehandlung



5.5.1 Expected Interrupt

```
CA_INTERRUPT:

        MOVL    @(SP)+  ,R4             ;GET POINTER TO IDB
        MOVL    IDB$L_OWNER(R4), R5     ;GET USER'S SPECIAL UCB
        MOVL    IDB$L_CSR(R4)   , R4    ;GET CSR ADDRESS

        DSBINT

        BSBW    CA_REGISTER                     ;SAVE REGISTERS

        BBCC    #UCB$V_INT, UCB$W_STS(R5),- ;EXPECTED INTERRUPT ?
                UNSOLC_INT                      ;NO. HANDLE UNSOLC INTERRUPT

        MOVW    #^O163777       ,CA_DMA_RES_CSR2(R4)
        CLRW    CA_DMA_CSR1(R4)                 ;CLEAR CSR1

        MOVQ    UCB$L_FR3(R5)   ,R3     ;RESTORE FORK CONTEXT

        PUSHR   #^M<R4>
        JSB     @UCB$L_FPC(R5)
        POPR    #^M<R4>

        JMP     RETURN

;       DISMISS INTERRUPT

DISMISS:BICW    #7      ,CA_CSR(R4)     ;CLEAR FUNCTION BITS IN CSR
        MOVB    #24     ,(R4)[RO]       ;CLEAR LAM WITH F10 A0

RETURN: BISW    #^O40   ,CA_CSR(R4)     ;ENABLE NEXT LAM FROM CAMAC

        ENBINT

        POPR    #^M<R0,R1,R2,R3,R4,R5>

        REI
```

5.5.2 Unsolicited Expected Interrupt

```
UNSOLC_INT:

        MOVZWL  UCB$W_SLOT(R5)  ,R0     ;SLOT ADDRESS IN R0
        TSTL    UCB$L_ASTLHD(R5)        ;ANY WAITING UNSOLC AST'S ?
        BEQL    20$                     ;NO, GO HIT UNSOLC LAM ON HEAD

;       DEFINED SLOT, LAM SET AND WAITING UNSOLC AST
;       ********************************************

        MOVL    R4      ,R1             ;SAVE CSR
        MOVAL UCB$L_ASTLHD(R5) ,R4      ;LAM WAS FOR THIS UNIT # SO

        JSB     G^COM$DELATTNAST        ;DELIVER UNSOLC AST'S

        MOVL    R1      ,R4             ;CSR TO R4
        MULW2   #CA$A_N1A0      ,R0     ;GET CSR OFFSET FOR SLOT
        BRB     DISMISS                 ;THEN DISMISS INTERRUPT
```

5.5.3 Unsolicited Unexpected Interrupt

```
;       DEFINED SLOT, LAM SET BUT NO WAITING UNSOLC AST
;       ***********************************************

20$:    MULL3   #CA$A_N1A0 ,R0 ,R1          ;REAL UNSOLICITED INTERRUPT
        ADDL3   #CA$A_N1A0-1 ,R1 ,R3        ;SHUT UP LAM WITH
30$:    MOVB    #24       ,(R4)[R1]         ;F24AX (DISABLE LAM) AND
        MOVB    #10       ,(R4)[R1]         ;F10AX (CLEAR LAM)
        ACBW    R3 ,#2 ,R1 ,30$             ;FOR A0 THRU A15

        BBC     #DEV$V_ELG,-                ;IS DEVICE ERROR LOGGING ENABLED ?
                UCB$L_DEVCHAR(R5),-         ;IF NOT RETURN, OTHERWISE LOG ERROR
                PILOT

        MOVL    R5        ,R3               ;SET UP UCB ADDR FOR ERRLOGGER
        PUSHAB  RETURN                      ;SET UP FORK PROCESS

        FORK

;       ****    FORK PROCESS TO LOG ERRORS

        MOVL    R3,       R5
        CLRB    UCB$B_ERRFLG(R5)            ;FLAG FOR UNSOLICITED INTERRUPTS

        JSB     G^ERL$DEVICERR

        ENBINT

        RSB
```

5.6 Post IO Routinen

5.6.1 Cancel IO

```
CA_CANCEL:                                  ;CANCEL I/O

        .IF DEFINED DISPLAY_INTERFACE
        CMPW    #20       ,UCB$W_SLOT(R5)           ;DISPLAY MODUL 20 ???
        BNEQ    2$                                  ;NO

;  DISABLE DISPLAY

        MOVW    #24       ,UCB$W_CAFUNC(R5)         ; FUNCTION = F24
        CLRW    UCB$W_SUBADR(R5)                    ; SUBADDR. = A0

        BSBW    EXE_FUNCTION                        ; DISABLE DISPLAY
        .ENDC


2$:
        TSTL    UCB$L_ASTLHD(R5)            ;ATTN AST ENABLED ?
        BEQL    20$                         ;NO
```

```
; FINISH ALL ATTN AST'S FOR THIS PROCESS.

        PUSHR   #^M<R2,R6,R7>
        MOVL    R2, R6                  ;SETUP CHANNEL NUMBER
        MOVAB   UCB$L_ASTLHD(R5), R7    ;ADDRESS OF LISTHEAD
        JSB     G^COM$FLUSHATTNS        ;FLUSH ATTN AST'S FOR PROCESS
        POPR    #^M<R2,R6,R7>

; CHECK TO SEE IF DATA TRANSFER REQUEST IS IN PROGRESS
; FOR THIS PROCESS ON THIS CHANNEL


20$:
        SETIPL  UCB$B_DIPL(R5)          ;LOCK OUT DEVICE INTERRUPTS
        JSB     G^IOC$CANCELIO          ;CHECK IF TRANSFER GOING
        BBC     #UCB$V_CANCEL, UCB$W_STS(R5), 30$
                                        ;BRANCH IF NOT FOR THIS GUY

; IF BLOCK MODE DMA REQUEST IN PROGRESS, RELEASE UBA RESOURCES
; IF TRANSFER IS IN PROGRESS, DO A DEVICE RESET

        PUSHR   #^M<R2,R3,R4>
        BBC     #UCB$V_INT, UCB$W_STS(R5), 25$
                                        ;BRANCH IF TRANSFER NOT IN PROGRESS
        MOVL    UCB$L_CRB(R5)   , R4    ;CALCULATE  C S R  ADDRESS
        MOVL    @CRB$L_INTD+VEC$L_IDB(R4), R4

        MOVW    #^O163777, CA_DMA_RES_CSR2(R4)  ;STOP DMA-MODUL
        CLRW    CA_DMA_CSR1(R4)                 ;CLEAR CSR1

        PURDPR                          ;PURGE UBA BUFFERED DATA PATH
        RELMPR                          ;RELEASE UBA MAP REGISTERS
        RELDPR                          ;RELEASE UBA DATA PATH REGISTER

25$:
        MOVZWL  UCB$W_SLOT(R5)  ,R0     ;SLOT ADDRESS IN R0
        MULL3   #CA$A_N1A0      ,R0, R1 ;REAL UNSOLICITED INTERRUPT
        ADDL3   #CA$A_N1A0-1    ,R1, R3 ;SHUT UP LAM WITH:
1$:     MOVB    #24     ,(R4)[R1]       ;F24AX (DISABLE LAM) AND
        MOVB    #10     ,(R4)[R1]       ;F10AX (CLEAR LAM)
        ACBW    R3, #2, R1, 1$          ;FOR A0 THRU A15

        POPR    #^M<R2,R3,R4>

        MOVZWL  #SS$_CANCEL, R0         ;STATUS IS REQUEST CANCELED
        CLRL    R1
        CLRL    UCB$L_ASTLHD(R5)        ;CLEAR AST ENTRY
        BICW    #<UCB$M_TIM!UCB$M_INT!UCB$M_TIMOUT!UCB$M_CANCEL!UCB$M_POWER>
                UCB$W_STS(R5)           ;CLEAR UNIT STATUS FLAGS

        REQCOM                          ;SYSTEM FINISH I/O

30$:
        SETIPL  UCB$B_FIPL(R5)          ;LOWER TO FORK IPL
        RSB
```

5.6.2 Register Dump

```
        .SBTTL  CA_REG_DUMP

;       CA_REG_DUMP, DUMPS DEVICE REGISTERS OR CAMAC PARAMETERS TO REGISTERS

;       FUNCTIONAL DESCRIPTION :

;               WRITE THE DEVICE REGISTERS AND/OR CAMAC PARAMETERS TO
;               ERROR BUFFER

;       INPUTS :

;               R0      - ADDRESS OF OUTPUT BUFFER
;               R4      - ADDRESS OF CSR
;               R5      - ADDRESS OF UCB
;               UCB$B_ERRFLG(R5)   - 0=UNSOLC'INT, 1=TIMEOUT, 2=POWERFAIL

;       OUTPUTS :

;               THE ROUTINE MUST PRESERE ALL REGISTERS EXCEPT R1 - R3
;               THE OUTPUT BUFFER CONTAINS THE DUMPED VALUES.
;               R0 CONTAINS THE ADDRESS OF THE NEXT EMPTY LONGWORD IN THE BU

CA_REG_DUMP:

        MOVZBL  #12.     ,(R0)+          ;LONG WORD COUNT FOR ERR MSG BUFR
        MOVZBL  UCB$B_ERRFLG(R5) ,R1     ;GET ERR LOG FLAG
        MNEGL   R1       ,(R0)+          ;SAVE NEG OF COUNT IN BUFR
        MOVZWL  CA_DATA_HIGH(R4)  ,(R0)+        ;SAVE DEVICE REGISTERS
        MOVZWL  CA_LAM_LOW(R4)    ,(R0)+
        MOVZWL  CA_LAM_HIGH(R4)   ,(R0)+
        MOVZWL  CA_CSR(R4)        ,(R0)+

20$:    TSTL    R1
        BNEQ    30$
        CLRQ    (R0)+                   ;FLAG=0 MEANS UNSOLC INTRPT
        MOVZWL  UCB$W_SLOT(R5)   ,(R0)+ ;SAVE SLOT NUMBER IN CRATE
        CLRQ    (R0)+                   ;THERE ARE NO QIO PARAMS TO SAVE
        BRB     90$
30$:    MOVZWL  UCB$W_CAFUNC(R5),(R0)+  ;SAVE QIO PARMS AS WELL
        MOVZWL  UCB$W_SUBADR(R5),(R0)+
        MOVZWL  UCB$W_SLOT(R5)   ,(R0)+ ;SAVE SLOT NUMBER IN CRATE
        MOVZWL  UCB$W_TIMEOUT(R5),(R0)+

90$:    RSB
```

6 Test - Beispiele

Bei der Driver Entwicklung wurde sehr viel Gewicht auf Test-Software gelegt. So entstanden nach und nach sehr flexible Testprogramme, die es ermoeglichen helfen bei einem Stoerfall jede Funktion auf der CAMAC Seite einzeln auszutesten.

6.1 Allgemeines Testprogramm

Das unten abgedruckte Programm ist ein Testprogramm von allgemeiner Anwendbarkeit. Nach dem Listing sind einige Anwendungsbeispiele aufgefuehrt.

```
C*************************************************************************
C
C       TASKNAME        :       CATE.FOR
C
C       BEARBEITER      :       H. HEER , H. STOFF
C
C       ERSTELLUNG      :       15-OCT-80
C
C       AENDERUNG       :       25-MAR-81
C
C       BETRIEBSSYSTEM  :       VAX11-780/VMS V2.1
C
C*************************************************************************
C
C       CATE IST EIN TESTPROGRAMM FUER DEN CAMAC - DRIVER / VAX 11 - VMS
C
        INTEGER *2 CRMO, CARE, CAWR, ISUB, IOSB(4), IERR(8), IFLAG
        INTEGER *2 CCBLK(4), ICRA, ISLOT
        INTEGER*4 STATUS, ICSR
        LOGICAL*1 A, B, C, D, E, F, G, H, Z, MODE, ENT
C
        EQUIVALENCE (IERR(1), IOSB(1))
        1           ,(IERR(5), STATUS)
        2           ,(IERR(7), IFLAG)
C
        DATA A/'A'/, B/'B'/, C/'C'/, D/'D'/, E/'E'/
        DATA F/'F'/, G/'G'/, H/'H'/, Z/'Z'/
        DATA CARE /0/, CAWR /16/
C
        EXTERNAL LAM_AST_SERV
```

```
C
C       ZUWEISUNG DES CAMAC GERAETE CRMO
C       ********************************
C
100     WRITE (6,501)
501     FORMAT ('$GIB MODUL-NUMMER : ')
        READ (6,502) ISLOT
502     FORMAT (1I2)
        WRITE (6,503)
503     FORMAT ('$GIB CRATE-NUMMER : ')
        READ (6,504) ICRA
504     FORMAT (1I1)
        WRITE (6,505)
505     FORMAT ('$GIB SUBADRESSE : ')
        READ (6,502) ISUB
C
        CALL CAMAC_ASSIGN (CRMO, ICRA, ISLOT, IERR)

1       WRITE (6,200)
200     FORMAT (' CAMAC - WRITE          = A'/
        1        ' CAMAC - WRITE & READ  = B'/
        2        ' CAMAC - FUNCTION TEST = C'/
        3        ' CSR - WRITE  ENA LAM  = D'/
        4        ' CSR - READ            = E'/
        5        ' SET LAM FROM DWD      = F'/
        6        ' RESET LAM FROM DWD    = G'/
        7        ' CONN UNSOLC INTERR.   = H'/

        8        ' TESTE ANDERES MODUL   : Z'//
        9        '$GIB TEST -MODE        : ')
C
        READ (5,300,END=1000) MODE
300     FORMAT (1A1)
        WRITE (6,201)
201     FORMAT (/)
C
        IF ((MODE .NE. A) .AND. (MODE .NE. B)) GO TO 400
C
        WRITE (6,202)
202     FORMAT ('$24 BIT IM LOOP ? (NEIN >>> CR) : ')
        READ (6,203) IG, ENT
203     FORMAT (Q,(A1))
C
        IF (IG .NE. 0) GO TO 204
C
        WRITE (6,205)
205     FORMAT ('$GIB DATE : ')
        READ (6,206) IDATO
206     FORMAT (I08)
C
        CALL CAMAC_WRITE (CRMO, ISUB, CAWR, IDATO, IERR)
C
        IF (MODE .NE. B) GO TO 2000
C
        CALL CAMAC_READ  (CRMO, ISUB, CARE, IDATI, IERR)
C
        IF (IDATO .EQ. IDATI) GO TO 208
C
        WRITE (6,207)IDATO, IDATI
207     FORMAT (' LESE - SCHREIB  FEHLER! IDATO ',O8.8,' IDATI ',O8.8)
C
208     GO TO 2000
```

```
204     IDATO = 1
C
        DO 10 I = 1,16777215
C
        IDATO = IDATO + 1
C
C       CAMAC - WRITE : SCHREIBE INS MODULE ISLOT
C       ****************************************
C
        CALL CAMAC_WRITE (CRMO, ISUB, CAWR, IDATO, IERR)
C
        IF (MODE .NE. B) GO TO 10
C
C       CAMAC - READ : LIES MODULE ISLOT
C       ********************************
C
        CALL CAMAC_READ (CRMO, ISUB, CARE, IDATI, IERR)
C
        IF (IDATO .EQ. IDATI) GO TO 10
C
        WRITE (6,207) IDATO, IDATI

10      CONTINUE
        GO TO 2000
C
400     IF (MODE .NE. C) GO TO 500
C
        WRITE (6,401)
401     FORMAT ('$ALLE FUNKTIONEN IM LOOP? (NEIN >>> CR) : ')
        READ (6,203) IQ, ENT
C
        IF (IQ .NE .0) GO TO 402
C
        WRITE (6,403)
403     FORMAT('$GIB FUNKTION : ')
        READ (6,404) IFUNC
404     FORMAT (1I4)
C
        CALL CAMAC_FUNCTION (CRMO, ISUB, IFUNC, IERR)
        GO TO 2000
C
402     IFUNC = -1
C
        DO 20 K = 1,10000
C
        IFUNC = IFUNC + 1
        IF (IFUNC .GT. 31) IFUNC = 0
        IF (IFUNC .EQ. 25) GO TO 20
C
C       SCHREIBE CAMAC FUNKTION INS MODUL CRMO
C       **************************************
C
        CALL CAMAC_FUNCTION (CRMO, ISUB, IFUNC, IERR)
C
20      CONTINUE
        GO TO 2000
C
500     IF (MODE .NE. D) GO TO 600
```

```
C       SCHREIBE CAMAC STATUS REGISTER - ENABLE LAM
C       ******************************************
C
        CCBLK(1) = '40'O
        CCBLK(2) = '17'O
        CCBLK(3) = '1444'O
        CCBLK(4) = '14'O
C
        CALL CAMAC_CSR_WRITE (CRMO, CCBLK, IERR)
        GO TO 2000
C
600     IF (MODE .NE. E) GO TO 700
C
C       LIES CAMAC STATUS REGISTER
C       **************************
C
        CALL CAMAC_CSR_READ (CRMO, CCBLK, IERR)
C
        WRITE (6,601) (CCBLK(I), I=1,4)
601     FORMAT (' CCBLK : ',4O6)

        GO TO 2000
C
700     IF (MODE .NE. F) GO TO 800
C
C       ENABLE LAM UND SET LAM IM MODUL CRMO
C       ************************************
C
        IFUNC = 24
C
        DO 30 I = 1,2
C
        IFUNC = IFUNC + 1
C
        CALL CAMAC_FUNCTION (CRMO, ISUB, IFUNC, IERR)
C
30      CONTINUE
        GO TO 2000
C
800     IF (MODE .NE. G) GO TO 900
C
C       RESET LAM VOM MODUL CRMO
C       ************************
C
        IFUNC = 10
C
        CALL CAMAC_FUNCTION (CRMO, ISUB, IFUNC, IERR)
        GO TO 2000
C
900     IF (MODE .NE. H) GO TO 910
C
C       ENABLE UNSOLICITED INTERRUPTS
C       *****************************
C
        CALL CAMAC_INTERRUPT (CRMO, LAM_AST_SERV, IERR)
C
910     IF (MODE .NE. Z) GO TO 920
C
        GO TO 100
920     GO TO 2000
```

```
$ RUN CATE
GIB MODUL-NUMMER : 18
GIB CRATE-NUMMER : 1
GIB SUBADRESSE : 0
CAMAC - WRITE          = A
CAMAC - WRITE & READ   = B
CAMAC - FUNCTION TEST  = C
CSR - WRITE  ENA LAM   = D
CSR - READ             = E
SET LAM FROM DWD       = F
RESET LAM FROM DWD     = G
CONN UNSOLC INTERR.    = H
TESTE ANDERES MODUL    : Z

GIB TEST -MODE         : B


24 BIT IM LOOP ? (NEIN >>> CR) :
GIB DATE : 1777
LESE - SCHREIB  FEHLER! IDATO 00001777 IDATI 00177777
I-O STATUS  :       1     4     1     1
S-S STATUS :         1
IFLAG       :     0

CAMAC - WRITE          = A
CAMAC - WRITE & READ   = B
CAMAC - FUNCTION TEST  = C
CSR - WRITE  ENA LAM   = D
CSR - READ             = E
SET LAM FROM DWD       = F
RESET LAM FROM DWD     = G
CONN UNSOLC INTERR.    = H
TESTE ANDERES MODUL    : Z

GIB TEST -MODE         : Z


GIB MODUL-NUMMER : 18
GIB CRATE-NUMMER : 2
GIB SUBADRESSE : 0
CAMAC - WRITE          = A
CAMAC - WRITE & READ   = B
CAMAC - FUNCTION TEST  = C
CSR - WRITE  ENA LAM   = D
CSR - READ             = E
SET LAM FROM DWD       = F
RESET LAM FROM DWD     = G
CONN UNSOLC INTERR.    = H
TESTE ANDERES MODUL    : Z

GIB TEST -MODE         : A


24 BIT IM LOOP ? (NEIN >>> CR) :
GIB DATE : 1777
I-O STATUS  :       1     4     0     1
S-S STATUS :         1
IFLAG       :     0
```

6.2 DMA Testprogramm

Unten stehendes Listing zeigt das Testprogramm fuer die DMA Hardware.

```
C*************************************************************************
C
C       NAME              :       DMA.FOR
C
C       BEARBEITER        :       H. HEER , H. STOFF
C
C       ERSTELLUNG        :       06-MAR-81
C
C       AENDERUNG         :       21-MAY-81
C
C       BETRIEBSSYSTEM    :       VAX11-780/VMS V2.1
C
C*************************************************************************
C
        INTEGER *2 DABU(4000), LABU, DMA_MODUL
        LOGICAL *1 IDAT(9), ITIM(8)
        INTEGER *2 CRMO, IERR(8)
        INTEGER *4 ICSR, LOOP
C
        DMA_MODUL = 4


        CALL DATE (IDAT)
        CALL TIME (ITIM)
C
        WRITE (6,10) IDAT, ITIM
10      FORMAT (' DMA-WRITE/READ - TESTPROGRAMM. START : ',9A1,4X,8A1//)
C
C       ZUWEISUNG DES CAMAC GERAETE-CRMOL
C       *********************************
C
        CALL CAMAC_ASSIGN (CRMO, 2, 18, IERR)
        IF (IERR(7) .EQ. -1) GO TO 1000
C
C       ENABLE LAM IM  D W D
C       ********************
C
        CALL CAMAC_FUNCTION (CRMO, 0, 26, IERR)
        IF (IERR(7) .EQ. -1) GO TO 1000
C
C       SETZE  L A M  IM  D W D
C       ***********************
C
        CALL CAMAC_FUNCTION (CRMO, 0, 25, IERR)
        IF (IERR(7) .EQ. -1) GO TO 1000
```

```
C
C       STOSSE  D M A  UEBERTRAGUNGEN AN
C       **********************************
C       **********************************
C
        IDA = 0
C
20      IDA   = IDA + 1

        IF (IDA .GT. '77777'O) THEN
        IDA = 1



        CALL TIME (ITIM)
        CALL DATE (IDAT)
        WRITE (6,4000) IDAT, ITIM
4000    FORMAT (' ***  32767 * 8K BYTES  UEBERTRAGEN  ***',9A1,4X,8A1)
        ENDIF


        DO 500  I = 1,  4000
500     DABU(I) = IDA


        CALL CAMAC_FUNCTION (CRMO, 0, 26, IERR)

        CALL CAMAC_DMA_WRITE (CRMO, DABU, 8000, 10, DMA_MODUL, IERR)
        IF (IERR(7) .EQ. -1) GO TO 1000
C
        WRITE (6,45) IERR(2), IERR(4), IERR(3)

        CALL CAMAC_FUNCTION (CRMO, 0, 26, IERR)


        CALL CAMAC_DMA_READ (CRMO, DABU, 8000, 10, DMA_MODUL, IERR)
        IF (IERR(7) .EQ. -1) GO TO 1000
C
        WRITE (6,45) IERR(2), IERR(4), IERR(3)

        GO TO 20



C       RESET  L A M  VOM  D W D
C       ************************
C
1000    CALL CAMAC_FUNCTION (CRMO, 0, 10, IERR)
        IF (IERR(7) .EQ. -1) GO TO 1000
C
```

```
C       DISABLE  L A M  VOM  D W D
C       **************************
C
        CALL CAMAC_FUNCTION (CRMO, 0, 24, IERR)
        IF (IERR(7) .EQ. -1) GO TO 1000
C

        WRITE (6,45) IERR(2), IERR(4), IERR(3)
45      FORMAT(' DMA-BCNT = ', I6, '   DMA-CSR1 = ', O6,
        1                          '   DMA-CSR2 = ', O6)
C

        DO 50 I=1, 4000, 160
50      WRITE (6, 60) I, (DABU(N), N=I, I+9)
60      FORMAT (' ',1I4,': ',(10I6))
C
        CALL DATE (IDAT)
        CALL TIME (ITIM)

        WRITE (6,30) IDAT, ITIM
30      FORMAT (' SO EIN TAG, ..... ',9A1,4X,8A1//)


C
        STOP
C
C
        END
```

6.3 Display Testprogramm

Fuer den Test der Display-Hardware dient unten abgedrucktes Testprogramm.

```
      INTEGER *2 DABU(32768)
      LOGICAL *1 IDAT(9), ITIM(8)
      CHARACTER  YES
      INTEGER *2 CRMO, IERR(8)
C
      CALL DATE (IDAT)
      CALL TIME (ITIM)
C
      WRITE (6,10) IDAT, ITIM
10    FORMAT (' CAMAC_DISPLAY - TESTPROGRAMM. START : ',9A1,4X,8A1//)
C
C     ZUWEISUNG DES CAMAC GERAETE-CRMOL
C     *********************************
C
      CALL CAMAC_ASSIGN (CRMO, 1, 20, IERR)
      IF (IERR(7) .EQ. -1) GO TO 1000
C
C
C
C     SCHREIBE EINE RAMPE IN DEN BUFFER
C     *********************************
C
      IA = 1
      IM = 8191
      IE = 16384


      DO 100  I = IA, IM
C
100   DABU(I) = I - 1
C
C
      DO 200  I = IM+1, IE

200   DABU(I) = IE + 1 - I

C
C     ENABLE CAMAC_DISPLAY FOR OUR BUFFER
C     ***********************************
C
      CALL CAMAC_DISPLAY (CRMO, DABU, IERR)
      IF (IERR(7) .EQ. -1) GO TO 1000
C
      WRITE (6, 30)
30    FORMAT('0***CR*** IF RETURN TO SYSTEM')
C
      READ  (5, 40) YES
40    FORMAT(A)
C
```

7 Datenraten

Die Messungen der Datenraten an einem Multi-User-System sind letzlich sehr relativ anzusehen, da starke Aktivitaeten der einzelnen Benutzer an der VAX die Umschaltzeiten des Betriebssystems betraechtlich veraendern koennen. Unsere Messungen koennen als Mittel der erreichbaren Uebertragungsgeschwindigkeiten angesehen werden.

7.1 24 bit Read - Write Zyklus auf ein CAMAC Modul

Die Transfergeschwindigkeit fuer den Schreibe bzw. Lesevorgang eines CAMAC Moduls ergab nach unseren Messungen 750 Read's oder Write's pro Sekunde.

Ein programmierter Blocktransfer entfiel, da die DMA Anschluesse sowohl fuer READ als auch fuer WRITE implementiert sind.

7.2 DMA Datenraten

Die Datenraten fuer den DMA Datentransfer werden von unserer CAMAC Hardware auf max 200000 16-bit Worte pro Sekunde begrenzt. Dies ist notwendig, wenn gleichzeitig noch weitere Geraete auf diesem Unibus DMA-Transfers machen muessen. Der gesamte Durchsatz auf einem Unibus der VAX 11/780 kann bis zu 650000 16-bit Worte pro Sekunde betragen.

8 Ausbaumoeglichkeiten

Der modulare Aufbau des Drivers erlaubt schnelle Aenderungen bzw. Erweiterungen der Driverfunktionen. So wurde beispielsweise die DMA Write Funktion nachtraeglich in den Driver implementiert.

8.1 DMA Doppelpufferbetrieb

Um hohe Zaehlraten bei statistisch anfallenden Ereignissen mit stark variierender Daterate im Blocktransfer effektiv zu gestalten, wurde der Doppelpuffer bzw. Wechselpufferbetrieb in das DMA Interface implementiert. Diese Moeglichkeit muesste ueber eine neue Driverfunktion in den CAMAC Driver eingebaut werden.

8.2 DMA Inkrementbetrieb.

Fuer die Verwendung eines PDP Rechners als Vielkanalanalysator wurde von der Hardware im DMA-Modul die Moeglichkeit geschaffen, jede adressierte Speicherzelle des angeschlossenen Rechners als Zaehler zu verwenden. Da in der Speichersteuerung der PDP selbs keine Moeglichkeit zur Inkrementierung vorgesehen ist, wird der Inhalt angewaehlter Adressen zunaechst in das DMA Modul transferriert um eins inkrementiert und dann zurueckgeschrieben. Auch diese Betriebsart liese sich ueber eine neue Funktion in den CAMAC Driver einbauen.

8.3 DMA Add to Memory

Fuer bestimmte Messaufgaben im physikalischen Bereich wurde die Moeglichkeit vorgesehen, den Inhalt der angewaehlten Adresse mit einem beliebigen Wert zu addieren, der von einem CAMAC Modul bereitgestellt wird.

Danksagung

Herrn W. John, der uns bei Hardwareaenderungen sowohl im CAMAC Crate-Controller und im DMA-Interface mit Rat und Tat hilfsbereit zur Seite stand.