VME Interface VC16

User Manual

CONTENT

14th January 1997

1. VME - CAMAC Interface VC16

1.1. General Description

The VC16 is a single-width 6U VME interface card for control of up to 15 CAMAC crates equipped with the W-Ie-Ne-R CC16 CAMAC crate controller.

The link between the CAMAC crates and the interface is performed by a 16-bit parallel interface (CP bus protocol) with a high speed data transfer (up to 1Mb/s). The cable length of the VC16 and CC16 link may be up to 200 metres (twisted pair cable) or up to 30m in case of standard flat cable. The VME interface module VC16 will serve for correct timing in case of cable length > 10 metres by stretching the VME cycles, i.e. delaying the acknowledge time from the slave connected.

The VC16 is an A24/D16 VME bus slave. The module supports the AM codes $39 and $3D. The VC16 sub-devices are addressed via 3 LSB address lines. The basic address is complete decoded and switch selectable. For LAM servicing one interrupt channel is installed. The interrupt request level (1-7) may be software selected in the VC16 status register.

The VC16 interface supports the special features of the CC16 like broadcast and block mode operation with automatic CAMAC NAF-code repeat. Additional the VC16 is equipped with an internal CP-bus cable delay correction and a CP-bus test feature .

The data transfer from / to the VME bus system has to be organised by an application program running on the VME CPU.

The different crates connected via the CC16 have 15 different geographic addresses and one broadcast address for parallel access. For special applications every single crate may be enabled / disabled for parallel download.

The VC 16 interface and the last CC16 controller in one chain have to be terminated to enable the correct function during the high speed data transfer.

1.2. Front Panel

Five status LED's are placed on the VC16 front panel to show ACCESS, LAM pending, LAM enable as well as write or mode operations of the CP bus protocol with the following functions,

ACCESS	-	lights if there is any access from the CPU
LAM pending	-	lights if there is a LAM request from the actual CAMAC crate
LAM enable	-	lights if the LAM interrupt line is enabled
write CP	-	lights if VC16 write to CP bus
read/ mode CP	-	lights if VC16 access to CP bus mode

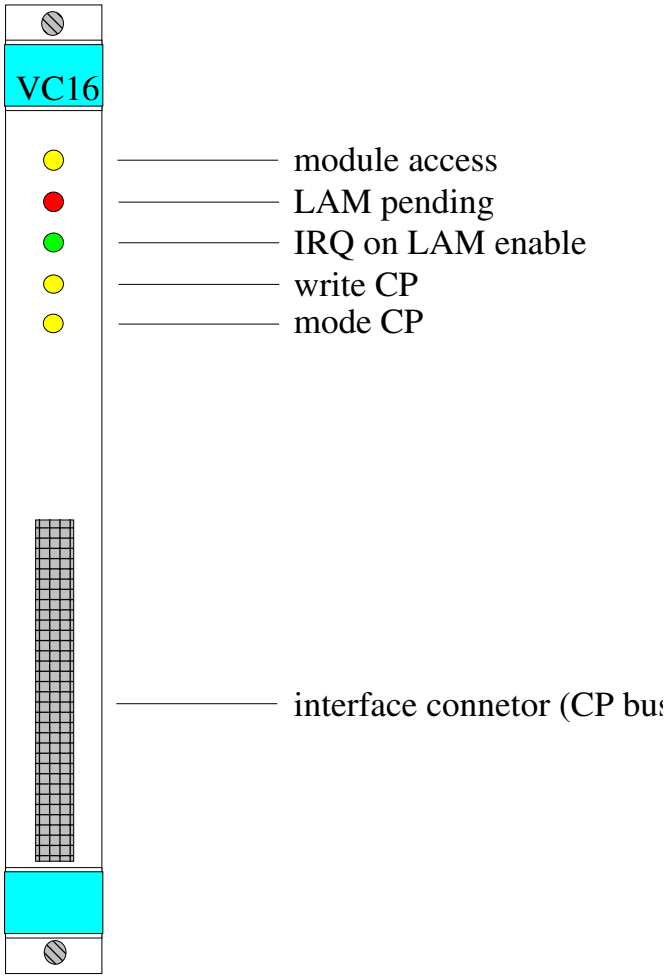


Fig. 1, CC16 front panel

Below the status LED's the 50-pin connector for parallel link to the interface (flat or twisted pair cable) is mounted Please note that both interface VC16 and the last CC16 in a chain have to be terminated by equal resistors (see next chapter)

1.3. Interface description

For any operation the CAMAC crate controller CC16 has to be linked to an external interface which is driven by a micro-processor or computer. The communication between the CC16 and the interface is performed according to the CP - "Communication bus Protocol" (RS422 - developed by RWTH Aachen). The CP bus consists of 16 bi-directional data lines and 7 control lines used for geographical addressing of the installed CC16 as well as for control and data transfer strobes. All signals are transferred as bipolar TTL signals.
Both the CC16 and the used interface have to be terminated to enable correct function during the high speed data transfer.

Fig. 2, VC16 block diagram

The crate controller termination is performed with resistor networks (RN6 - RN8). Two types of cable termination are provided:

type	resistor	cable length
short distance	DIL 220 Ohm	<30m
long distance*	2 x SIL8 + 1 x 100Ohm *	>30m

Table 1 Termination resistor types (* factory prepared)

Both ends of the link, i.e. CC16 and interface have to be terminated with equal resistors. Note, that in case of multi-crate systems only the last CC16 in the chain has to be terminated.

Maximum tested distances between crate controller and interface are 30 metres in case of 50 pin-flat cable or up to 200 metres using twisted pair cable.

The pin assignment of the 50-pin CP-bus interface connector is given in the following table.

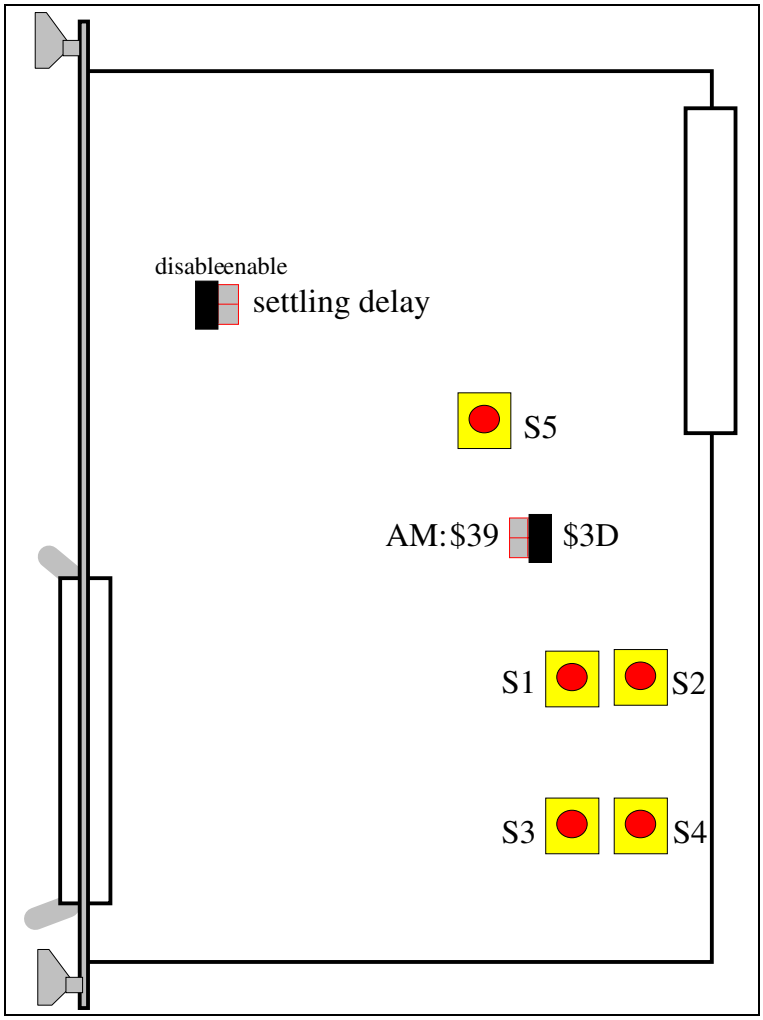
Pin	Line	Pin	Line
01	D0*	26	D12
02	D0	27	D13*
03	D1*	28	D13
04	D1	29	D14*
05	D2*	30	D14
06	D2	31	D15*
07	D3*	32	D15
08	D3	33	GND
09	D4*	34	LAM
10	D4	35	STATUS*
11	D5*	36	AKN
12	D5	37	BusN1*
13	D6*	38	BusN1
14	D6	39	BusN2*
15	D7*	40	BusN2
16	D7	41	BusN4*
17	D8*	42	BusN4
18	D8	43	BusN8*
19	D9*	44	BusN8
20	D9	45	BusMODE*
21	D10*	46	BusMODE
22	D10	47	BusWRITE*
23	D11*	48	BusWRITE
24	D11	49	BusS*
25	D12*	50	BusS

Table 2, CP-bus Interface connector pin assignment

1.4. VC16 installation

Prepare the VC16 before inserting into the 6U-VME system, i.e. check or select the register range by the hexadecimal switches (S1 - S5 according to the following scheme and example for BADR = $800000).

Switch	number	1	2	3	4	5	-
	value	8	0	0	0	0	-
Address	hex.	8	0	0	0	0	0
	bin.	0100	0000	0000	0000	0000	0000



Further the address modifier ($39 or $3D) has to be chosen by jumper setting on the VC16 board. The settling delay jumper should be in case of standard cable lengths up to 20 metres in the disabel position.

Switch off the VME crate and insert the VC16 into an empty VME slot. Connect the CC16 chain with the VC 16 by the 50 pin flat cable and switch on VME crate mains. For correct operation the VC16/CC16 system has to be initialised first by setting the geographical address of the called controller. Further the initial conditions for inhibit line (I - set off) and IRQ on LAM should be defined (see next chapters). At the end of initialisation a CAMAC initialise (Z) should be performed.

2. Register Layout and operating modes

The access to the CAMAC crate controller is performed according to the CP bus protocol via calls of several modes which are for selection of read / write operation or special controller functions. These modes are primary modes (**PM**) defining the type of operation and sub-modes (**SM**) for the controller functions (detailed description see CC16 Manual). In opposite to the personal computer interface PC16 with the VME VC16 interface there is no more a direct access to the primary modes. The primary mode selection is automatically done during sub-mode write as well as write and read data by using several registers. Also for the geographical address load a single register is used.

The complete 14 byte register block is organised according to the following scheme:

Name	Size	Address	Access type
Status Register	D0 - D11	BASE + $0	read / write
Mode Register	D0 - D4	BASE + $2	read / write
Output Data	D0 - D15	BASE + $4	read / write
Input Data	D0 - D15	BASE + $6	read
Geogr. Address	D0 - D7	BASE + $8	read / write
Interrupt Vector	D0 - D7	BASE + $A	read / write
Set Test Interrupt	dummy	BASE + $C	write

Table 3, VC16 address layout

The **Status register** ***(BASE+$0)*** includes the VME interrupt levels, the LAM and CP status as well as the cable delay information which is necessary for correct timing in a high speed data transfer with long distances between the CAMAC crates and the interface. The bit assignment is given in the following table:

Bit	Description	Access
0	interrupt level 1	read / write
1	interrupt level 2	read / write
2	interrupt level 4	read / write
3	0=disable, 1=enable LAM interrupt	read / write
4	0=LAM pending, 1=no LAM	read
5	CP acknowledge - CAMAC X	read
6	CP status - CAMAC Q	read
7	test interrupt	read
8	cable delay 1 (20m length)	read / write
9	cable delay 2 (30m length)	read / write
10	cable delay 4 (50m length)	read / write
11	cable delay 8 (100m length)	read / write

Table 4, VC16 Status register bit assignment

To test the correct connection between CAMAC crate and interface the VC16 is equipped with a read back mode register facility which can be red after a CP-bus activity. This feature can be used for troubleshooting in case of link fail functions. The last called primary mode is stored in bit D0 and D1. The next three bits show the type of the CP bus activity. The detailed read back register pin assignment is given in the following table,

Bit	Description	Example		
		write sub-mode	write data	read data
1	primary mode	1	0	1
2	primary mode	0	1	1
3	CP bus strobe	1	1	1
4	CP bus mode	1	0	0
5	CP bus write	1	1	0
	Read back value	$1D	$16	$7

Table 5, CP-bus read back bit description

The primary modes of the CP bus protocol which are indicated by the first two bits are listed in the following table,

primary mode	function
1	set geographical address
2	select device sub-mode register
3	write data
4	read data.

Table 6, CP-bus primary modes

SM	hex-value.	Function
0	0	write N, F, A and prompt CAMAC cycle bit
1	1	write high byte (8 bit) to CAMAC dataway
2	2	write low word (16 bit) to CAMAC dataway
3	3	read low word (16 bit) from CAMAC dataway
4	4	read high byte (8 bit) from CAMAC dataway
5	5	block mode read low word (16 bit) with NAF-repeat
6	6	CAMAC cycle with C or Z if enabled
7	7	enable for read / write CC16 status register (SCB)
8	8	enable CAMAC-C for next SM6 call
9	9	enable CAMAC-Z for next SM6 call
10	A	set CAMAC-I to true
11	B	set CAMAC-I to false
12	C	enable VC16 LAM servicing
13	D	disable VC16 LAM servicing

Table 7, CC16 Sub-modes

Via the 4 bits of the **Mode register** ***(BASE+$2)*** the different sub-modes of the CAMAC crate controllers are called. These sub-modes (SM) which chooses the controller function for CAMAC access are defined as follows:

The **Output Data register** ***(BASE+$4)*** as well as the **Input Data register** ***(BASE+$6)*** with a length of 16 bit are for the data transfer from and to the connected CAMAC crate controllers for:

- NAF-code (16 bit word, only output)
- low data word (16 bit)
- high data byte (8 bit).

As described in more detail in the CC16 Manual the NAF-code includes the station number N of an addressed module, the sub-address A and the function number F. It is decoded according to the following bit assignment:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N1	N2	N4	N8	N16	F1	F2	F4	F8	F16	A1	A2	A4	A8	--	S

Table 8, NAF code bits

The last bit S corresponds to an immediate CAMAC cycle on a NAF-load. If this bit is 0 the NAF load in the CC16 will not cause a prompt CAMAC cycle. S=0 is necessary write data to the dataway. For read-operations one has to set S=1.

The **Geographic Address register** ***(BASE+$8)*** is for selecting the CAMAC crate for the following operations. The chosen number defined by the first 4 bits ($0 ... $E) has to be equal to the value of the address switch on the CC16 front panel which has to be accessed. The address $F is used for broadcast operations to all connected controllers. For test purposes the geographic address may be read back.

The VC16 is prepared for interrupt based LAM servicing. The interrupt request level IRQ1* ... IRQ7* is defined by the setting of the first three bits of the VC16 status register. The 8-bit STATUS/ID for the interrupt acknowledge cycle [D08(O)] has to be load into the **Vector register** ***(BASE+$A)***. To test the interrupt function via the **Test Interrupt register** ***(BASE+$C)*** a software interrupt may be produced. Note, that for this operation the LAM enable bit in the status register must be set to true. The interrupt reset is done by clearing the LAM enable bit.

3. Programming and CAMAC operation

The VC16 / CC16 operation is performed on the basis of setting or reading the several VC16 registers which are described in chapter 2. The following instruction schemes are given in terms of setting these modes and registers and writing / reading data. Please consult the CC16 manual for the detailed CC16 functions.

3.1. Call and set crate on/off

select crate, write geographical address

`write Geo.Addr.register`*`(BASE+$8)`* - set crate / branch

Set crate on / off

`set SM7`*`(BASE+$2)`* - set SM to CC16 status register
`write SCB`*`(BASE+$4)`* - write scb with: bit2=0 switch crate on
bit2=1 switch crate off

to test the setting read the SCB with

`read SCB`*`(BASE+$8)`* - read SCB with: bit2=0 crate is on
bit2=1 crate is off

The crate ON / OFF is displayed by a status LED at the front panel of the CC16. These ON / OFF modes are only relevant for broadcast operations.

3.2. General CAMAC commands Z, C, I

CAMAC INITIALISE - Z

```
set SM9(BASE+$2)        - set SM to enable Z
set SM6(BASE+$2)        - set SM to perform Z cycle
```

Q and X response will be set to 1 as response of CC16 to the Z-cycle.

CAMAC CLEAR - C

```
set SM8(BASE+$2)        - set SM to enable C
set SM6(BASE+$2)        - set SM to perform C cycle
```

Q and X response will be set to 1 as response of CC16 to the C-cycle.

SET CAMAC INHIBIT - I

```
set SM10(BASE+$2)       - set SM to enable I
```

SET BACK CAMAC INHIBIT - I

```
set SM11(BASE+$2)       - set SM to disable I
```

The status of the inhibit line (I) is displayed at the CC16 front panel by the Inhibit LED.

3.3. LAM servicing

ENABLE IRQ ON LAM REQUEST

```
set SM12(BASE+$2)       - set SM to enable IRQ on LAM request
```

DISABLE IRQ ON LAM REQUEST

```
set SM13(BASE+$2)       - set SM to disable IRQ on LAM request
```

READ STATUS REGISTER SCB

```
set SM7(BASE+$2)        - set SM to CC16 status register
read data(BASE+$6)      - read SCB (bits see CC16 manual)
```

CC16 INITIALISATION

The complete CC16 initialisation should include setting of crate / branch and SCB as well as of giving Z-cycle and checking for no setting of I-line,

```
write Geo.Addr.register(BASE+$8) - set crate / branch
set SM7(BASE+$2)        - set SM to CC16 status register
write SCB(BASE+$4)      - write scb with bit2=0 switch crate on
```

3.4. Read and write data

ACCESS AND DATA TRANSFER TO CAMAC MODULES

The data transfer to or from modules of the CC16 controlled crate has to be done in two steps due to the 24-bit word length of the CAMAC bus. First read or write the low and middle byte as one 16-bit word. The second cycle is for reading / writing the high byte if

required. However a lot of CAMAC modules use only word length up to 16 bit. In this case 16-bit data transfer is faster and more useful.

WRITE 24-BIT WORD

```
set SM0(BASE+$2)        - set SM to write NAF
write NAFBASE+$4)       - write NAF -word with S-bit=0
set SM1(BASE+$2)        - set SM to write high byte
write data(BASE+$4)     - write high byte
set SM2(BASE+$2)        - set SM to write low word
write data(BASE+$4)     - write low word (0-15) and start CAMAC
```
cycle

WRITE 16-BIT WORD

```
set SM0(BASE+$2)        - set SM to write NAF
write NAFBASE+$4)       - write NAF -word with S-bit=0
set SM2(BASE+$2)        - set SM to write low word
write data(BASE+$4)     - write low word (0-15) and start CAMAC
```
cycle

READ 24-BIT WORD

```
set SM0(BASE+$2)        - set SM to write NAF
write NAFBASE+$4)       - write NAF -word with S-bit=1 and start
                          CAMAC cycle
set SM3(BASE+$2)        - set SM to read low word
read data(BASE+$6)      - read low word (0-15)
set SM4(BASE+$2)        - set SM to read high byte
read data(BASE+$6)      - read high byte
```

READ 16-BIT WORD

```
set SM0(BASE+$2)        - set SM to write NAF
write NAFBASE+$4)       - write NAF -word with S-bit=1 and start
                          CAMAC cycle
set SM3(BASE+$2)        - set SM to read low word
read data(BASE+$6)      - read low word (0-15)
```

16-BIT FAST BLOCK TRANSFER (READ)

```
set SM0(BASE+$2)        - set SM to write NAF
write NAFBASE+$4)       - write NAF -word with S-bit=1 and start
                          CAMAC cycle
set SM5(BASE+$2)        - set SM to block read low word
read data(BASE+$6)      - read low word (0-15)
read data(BASE+$6)      - read low word (0-15)
read data(BASE+$6)      - read low word (0-15)
read data(BASE+$6)      - read low word (0-15)
read data(BASE+$6)      - read low word (0-15)
  ...
```
the last word has to be read again by the full sequence :

```
set SM3(BASE+$2)        - set SM to read low word
read data(BASE+$6)      - read low word (0-15)
```

3.5. CAMAC status bits Q and X

The status bits Q and X are stored in the status register of the interface (**ISR**) in the first bits CSD0 and CSD1. The Q- and X- bits will be stable within 1.4µs.

`read ISR` - read ISR Q = bit 6 X = bit 5

3.6. Cable delay correction

The VC16 will serve for correct timing in case of cable lengths by strectching the VME bus cycles, e.g. delaying the acknowledge time from the controllers connected. The delay may be different for the 15 geographical addresses. The several delay values are defined within bit 8-11 of the VC16 status register. The following table gives examples for typical cable lengths / delay times (including CAMAC cycle),

cable length	delay time	status register (hex.)	bit assignment (bin.)
<10m	1.2µs	x0xx	xxxx 0000 xxxx xxxx
<30m	1.4µs	x1xx	xxxx 1001 xxxx xxxx
<50m	1.8µs	x3xx	xxxx 0011 xxxx xxxx
<100m	2.8µs	x7xx	xxxx 0111 xxxx xxxx
<200m	5.2µs	xFxx	xxxx 1111 xxxx xxxx

Table 9, Interface cable delay programming

To fix the cable delay first select the geographical crate address. Then the bits 8-11 of the statusregister has to be defined.

write Geo.Addr.reg.*(BASE+$8)* - set crate / branch
write Status register *(BASE)* - write status register

3.7. Using interrupts

The VC16 is able to generate an interrupt on the VME bus after receiving a LAM request. The interrupt level is defined within the VC16 status register, i.e. bits 0 - 2 (see table 4). The vector address is loaded into the VC16 vector register (BASE+$A, see table 3).

For interrupt based LAM servicing first the VC16 has to be enabled for interrupt on LAM. This is done by setting bit 3 of the VC16 status register to 1. To reset the LAM interrupt disable, i.e. set bit 3 to 0.

The interrupt servicing on the VME bus which is strongly CPU dependant has to be defined within the CAMAC control and read out software. For testing the interrupt can be generated by writing any dummy value into the test interrupt register (BASE+$C). For interrupt programming see the VME-CPU manual and programming language description.

4. C-Library and software examples

The following chapter describes the contents of the OS-9 software kit. The included C-code library for the VC16 / CC16 is based on a general definition of the VME register access by the commands *get_reg(address)* and *put_reg(address, data)*. This access which depends on the chosen address modifier AM, i.e. on the size of the address range (16bit /24bit/32bit) as well as on the kind of operation (supervisor / data / user / ...). is strongly CPU dependent. Thus in the unit **reg_acc.c** the register access operations get_reg and put_reg used in the **cam_vc16.c** library has to be defined for the used VME bus master. Programming examples are given for the ELTEC E7/E8 and the CES FIC8232 modules.
As an example for using the C-library please find the test program **vc16test.c**.

```
/*
 cam_vc16.c
 ***************************************************************************
                    CC16 / VC 16 C routines
  ------------------------------------------------------------
   Unit contains :
     cam_adr                    - set I/O address (BADR)
     getstatus                  - get CC16 status
     cam_cratecheck             - test crate available
     set_crate                  - set geographical crate address
     set_delay                  - set value for cable delay (4 bits)
     cam_controller_ini         - controller initialization
     cam_z                      - CAMAC Initialize (Z)
     cam_c                      - CAMAC Clear (C)
     cam_i                      - CAMAC Inhibit (I)
     cam_ci                     - CAMAC Clear + Inhibit (I) setzen
     cam_irq                    - enable CAMAC LAM-Interrupt
     cam_noirq                  - disable CAMAC LAM-Interrupt
     cam_q                      - get CAMAC Q-response
     cam_x                      - get CAMAC X-response
     cam_nfa                    - set N,F,A start CAMAC cycle
     cam_nfa_read               - set N,F,A and read 2 byte
     cam_nfaqx_read             - set N,F,A and read 2 byte with Q and X
     cam_nfaqx_read24           - set N,F,A and read 3 byte with Q and X
     cam_nfa_write              - set N,F,A and write 2 byte
     cam_nfaqx_write            - set N,F,A and write 2 byte with Q and X
     cam_nfaqx_write24          - set N,F,A and write 2 byte with Q and X
     cam_lam                    - get LAM source
     cam_lamf                   - any LAM request

     badr := basic I/O address
     copyright:    A. Ruben, W-Ie-Ne-R, PleinBaus GmbH  20.03.95

 ***************************************************************************
 CPU dependent register access has to be defined in unit reg_acc.h
*/
#include "reg_acc.c"

unsigned int nfa;
unsigned long int badr;
unsigned long int badr2;
unsigned long int badr4;
unsigned long int badr6;
unsigned long int badr8;
unsigned long int badrA;
unsigned long int badrC;
```

```
void cam_adr(unsigned long int);
int  cam_cratecheck (void);
void set_crate (int);
void set_delay (int);
void cam_controller_ini (unsigned long int);
int  getstatus(void);
void cam_z(void);
void cam_c(void);
void cam_i(void);
void cam_ci(void);
void cam_0(void);
int  cam_q(void);
int  cam_x(void);
void cam_irq(void);
void cam_noirq(void);
void cam_nfa(int, int, int);
void cam_nfa_write(int, int, int, unsigned int);
void cam_nfaqx_write(int, int, int, unsigned int, int *, int *);
void cam_nfaqx_write24(int, int, int, long int, int *, int *);
unsigned int  cam_nfa_read(int, int, int);
unsigned int  cam_nfaqx_read(int, int, int, int *, int *);
long int cam_nfaqx_read24(int, int, int, int *, int *);

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

void cam_adr(unsigned long int bad)
{
 badr=bad;
 badr2=badr+2;
 badr4=badr+4;
 badr6=badr+6;
 badr8=badr+8;
 badrA=badr+10;
 badrC=badr+12;
};
/*****************************************************************************/

int  cam_cratecheck (void)
{
 int j;
 put_reg(badr8,0);
 put_reg(badr2,7);
 put_reg(badr4,0);
 j=get_reg(badr6);
 j=(j & 4) - 4;
 if (j<0)
 {
  j=1;
 }
 return(j);
};
/*****************************************************************************/

void  set_crate (int crate)
{
 put_reg(badr8,crate);
};
/*****************************************************************************/
```

```
void set_delay (int delay_bits)
{
 unsigned int i;
 i=get_reg(badr);
 i=i & 0x0ff;
 i=i | (delay_bits << 8);
 put_reg(badr,i);
}
/*************************************************************************/

void cam_controller_ini (unsigned long int bad)
{
 vme_init();
 cam_adr(bad);
 set_crate(0);
 cam_z();
};
/*************************************************************************/

int getstatus(void)
{
 char j;
 j=0;
 return(j & 7);
};
/*************************************************************************/

void cam_z(void)
{
 put_reg(badr2,11);
 put_reg(badr2,9);
 put_reg(badr2,6);
};
/*************************************************************************/

void cam_c(void)
{
 put_reg(badr2,11);
 put_reg(badr2,8);
 put_reg(badr2,6);
};
/*************************************************************************/

void cam_i(void)
{
 put_reg(badr2,10);
};
/*************************************************************************/

void cam_ci(void)
{
 put_reg(badr2,10);
 put_reg(badr2,8);
 put_reg(badr2,6);
};
/*************************************************************************/

int cam_q(void)
```

```
{
 char j;
 j=get_reg(badr);
 return ( j >> 6) & 1;
};
/*****************************************************************************/

int cam_x(void)
{
 char j;
 j=get_reg(badr);
 return ( j >> 5) & 1;
};
/*****************************************************************************/

void cam_irq(void)
{
 put_reg(badr2,12);
};
/*****************************************************************************/

void cam_noirq(void)
{
 put_reg(badr2,13);
};

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

void cam_nfa(int n, int f, int a)
{
 nfa= a << 5;
 nfa= (nfa | f) << 5;
 nfa= nfa | (n+0x8000);
 put_reg(badr2,0);
 put_reg(badr4,nfa);
};
/*****************************************************************************/

void cam_nfa_write(int n, int f, int a, unsigned int data)
{
 nfa= a << 5;
 nfa= (nfa | f) << 5;
 nfa= nfa | n;
 put_reg(badr2,0);
 put_reg(badr4,nfa);
 put_reg(badr2,2);
 put_reg(badr4,data);
};
/*****************************************************************************/

void cam_nfaqx_write(int n, int f, int a, unsigned int data,int *q, int *x)
{
 char j;
 nfa= a << 5;
 nfa= (nfa | f) << 5;
 nfa= nfa | n;
 put_reg(badr2,0);
 put_reg(badr4,nfa);
 put_reg(badr2,2);
```

```
 put_reg(badr4,data);
 j=get_reg(badr);
 *q=( j >> 6 ) & 1;
 *x=( j >> 5 ) & 1;
};
/*************************************************************************/

void cam_nfaqx_write24(int n, int f, int a, long int data,int *q, int *x)
{
 char j;
 int ih;
 int il;
 ih= (int) ((data >> 16) & 255);
 il= (int) data;
 nfa= a << 5;
 nfa= (nfa | f) << 5;
 nfa= nfa | n;
 put_reg(badr2,0);
 put_reg(badr4,nfa);
 put_reg(badr2,1);
 put_reg(badr4,ih);
 put_reg(badr2,2);
 put_reg(badr4,il);
 j=get_reg(badr);
 *q=( j >> 6 ) & 1;
 *x=( j >> 5 ) & 1;
};
/*************************************************************************/

unsigned int cam_nfa_read(int n, int f, int a)
{
 nfa= a << 5;
 nfa= (nfa | f) << 5;
 nfa= nfa | (n+0x8000);
 put_reg(badr2,0);
 put_reg(badr4,nfa);
 put_reg(badr2,3);
 return ((unsigned int) get_reg(badr6));
};
/*************************************************************************/

unsigned int cam_nfaqx_read(int n, int f, int a, int *q, int *x)
{
 char j;
 nfa= a << 5;
 nfa= (nfa | f) << 5;
 nfa= nfa | (n+0x8000);
 put_reg(badr2,0);
 put_reg(badr4,nfa);
 put_reg(badr2,3);
 nfa= (unsigned int) get_reg(badr6);
 put_reg(badr,3);
 nfa=(unsigned int) get_reg(badr4);
 j=get_reg(badr);
 *q=( j >> 6 ) & 1;
 *x=( j >> 5 ) & 1;
 return (nfa);
};
/*************************************************************************/
```

```
long int cam_nfaqx_read24(int n, int f, int a, int *q, int *x)
{
 char j;
 long int lh;
 nfa= a << 5;
 nfa= (nfa | f) << 5;
 nfa= nfa | (n+0x8000);
 put_reg(badr2,0);
 put_reg(badr4,nfa);
 put_reg(badr2,4);
 lh=get_reg(badr6) & 255;
 lh=lh <<16;
 put_reg(badr2,3);
 lh += get_reg(badr6) & 0xffff;
 j=get_reg(badr);
 *q=( j >> 6 ) & 1;
 *x=( j >> 5 ) & 1;
 return (lh);
};
/*****************************************************************************/

int cam_lam(int station)
{
 cam_nfa(station,8,0);
 return (cam_q());
};
/*****************************************************************************/

int cam_lamf(void)
{
 int j;
 j=get_reg(badr);
 return(!j & 2);
};
```

```
/* reg_acc.c
  ***********************************************************************

                        VME-bus CPU register Access

    copyright:    W-Ie-Ne-R, PleinBaus GmbH, A. Ruben  20.12.94

 ***********************************************************************

void vme_init(void);
void put_reg(unsigned long int ,unsigned long int );
unsigned long int get_reg(unsigned long int);

#define vc16_base   0x800000            /* for VC16 rotary switches                 */
#define eltec_base  0xff000000          /* for ELTEC CPU, AM$39                     */
#define cesfic_base 0xf0000000          /* for CES FIC8232 CPU, 16bit data, AM$39*/

#define am 0x39                         /* Address modifier for ATVME system    */
                                        /* jumper to AM$39 on VC16 board        */

/*
################  ELTEC 68X00 CPU System  ################
  AM $39 by address offset
*/
#ifdef ELTEC

void put_reg(unsigned long int regadr,unsigned long int data)
{
        regadr+=eltec_base;
        * ( short * ) regadr = data;
}

unsigned long int get_reg(unsigned long int regadr)
{
        regadr+=eltec_base;
        return  ( * ( short * ) regadr);
}

void vme_init(void)
{
}
#endif

/*
################  CES FIC8232 CPU System  ################
 AM $39 by address offset
*/
#ifdef CES


void put_reg(unsigned long int regadr,unsigned long int data)
{
        regadr+=cesfic_base;
        * ( short * ) regadr = data;
}

unsigned long int get_reg(unsigned long int regadr)
{
        regadr+=cesfic_base;
```

```
        return  ( * ( short * ) regadr);
}

void vme_init(void)
{
}
#endif

/*
 ################  PRO_VME PC-VME System ############
*/

#ifdef ATVME
#include "VMELIB.C"

void put_reg(unsigned long int regadr,unsigned long int data)
{
        WriteWordExt(regadr,data,am);
}

unsigned long int get_reg(unsigned long int regadr)
{
        return ReadWordExt(regadr, am);
}

void vme_init(void)
{
        Init_AT_VME();
}
#endif
```

```
/*
  W-Ie-Ne-R CC16/VC16 Test and Demonstration
            Version 1.1, revised 4/95
  copyright,  WIENER - Plein & Baus GmbH, 04.04.95
*/

#include <stdio.h>
#include "cam_vc16.c"

#define vc16_base   0x800000    /* for VC16 rotary switches       */
#define cab_delay   0x0         /* for VC16-cc16 link cable delay */

int main()
{
  int   n1,f1,a1;
  int   q,x;
  long int   data;
  char  inbuf [130];
  cam_controller_ini(vc16_base);
  n1=1;
  printf("\n           W-IE-NE-R CC16/VC16 DEMONSTRATION  \n");
  printf("\n   ---------------------------------------------------- \n \n");
  printf("\n NAF code operation demonstration (N=0 to stop) \n");
  printf("\n copyright Plein & Baus GmbH, Burscheid/Germany - VI/94 \n \n");

/* demonstration of clear, inhibit and inizialise  */

  cam_c();
  cam_i();
  cam_z();
  set_delay(0x0);

/* demonstration of NAF read and write operations   */

cam_irq();

  while (n1 != 0)
  {
   printf("\n   ----------------------------------------------------  \n");
   printf("\n CAMAC station number   N=  \t");
   gets(inbuf);
   sscanf(inbuf, "\t%d", &n1);
   printf("\n CAMAC function number  F= \t");
   gets(inbuf);
   sscanf(inbuf, "\t%d", &f1);
   printf("\n CAMAC subaddress       A= \t");
   gets(inbuf);
   sscanf(inbuf, "\t%d", &a1);
   printf("\n");
   if (f1>8)
   {
   printf(" data to write : \t");
   gets(inbuf);
   sscanf(inbuf, "\t%li", &data);
   cam_nfaqx_write24(n1,f1,a1,data,&q,&x);
   printf("\t\t\t  Q= %i    X= %i",q,x);
   }
```

```
    else
    {
    data = cam_nfaqx_read24(n1,f1,a1,&q,&x);
    printf(" data= %li",data);
    printf("\t Q= %i    X= %i \n",q,x);
    }
  }
}
```

Software Licence Agreement

Program: VC16 libraries and test/demonstration program (OS-9 or IBM-PC version)

We do not make representations or warranties to the contents or use of the program and the software description.

Further, we reserve the right to make changes to any and all parts of the software and the manual, at any time, without obligation to notify any person or entity of such changes.

W-Ie-Ne-R, Plein & Baus GmbH
Müllersbaum 20
D - 51399 Burscheid
Germany

Phone: (49) (0) 2174 - 678 0
FAX: (49) (0) 2174 - 678 555
E-Mail: info@wiener-d.com
URL: http://www.wiener-d.com