CC-USB

User Manual

WIENER, Plein & Baus GmbH
Revision 5.02, July 6, 2010

www.wiener-d.com

General Remarks

The only purpose of this manual is a description of the product. It must not be interpreted a declaration of conformity for this product including the product and software.
W-Ie-Ne-R revises this product and manual without notice. Differences of the description in manual and product are possible.
W-Ie-Ne-R excludes completely any liability for loss of profits, loss of business, loss of use or data, interrupt of business, or for indirect, special incidental, or consequential damages of any kind, even if **W-Ie-Ne-R** has been advises of the possibility of such damages arising from any defect or error in this manual or product.
Any use of the product which may influence health of human beings requires the express written permission of **W-Ie-Ne-R**.
Products mentioned in this manual are mentioned for identification purposes only. Product names appearing in this manual may or may not be registered trademarks or copyrights of their respective companies.
No part of this product, including the product and the software may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means with the express written permission of **W-Ie-Ne-R**.

CC-USB and VM-USB are designed by JTEC Instruments.

CC-USB FIRMWARE Version 5.02

Table of contents:

1 GENERAL DESCRIPTION

The CC-USB is a full-featured CAMAC Crate controller with integrated high speed USB interface. It supports Master and Slave operations with full CAMAC arbitration; as a master it accepts slaves. The CC-USB is FASTCAMAC compliant. The CC-USB internal FPGA can be programmed to operate as command sequencer with data buffering in a 22kB FIFO. Combined with front panel triggering via the CAMAC operation and data taking can be done without any PC or USB activity.
All CC-USB logic is controlled by the XILINX Spartan 3 FPGA. Upon power-up the FPGA boots from a flash memory. The configuration flash memory can be reprogrammed via the USB port, allowing convenient updates of the firmware. 4 memory sections allow upload and use of different firmware versions.
The integrated CAMAC data way display as well as additional user and status LED's for the controller and the USB port provide all necessary system information for monitoring, hardware control and debugging.

1.1 CC-USB Features

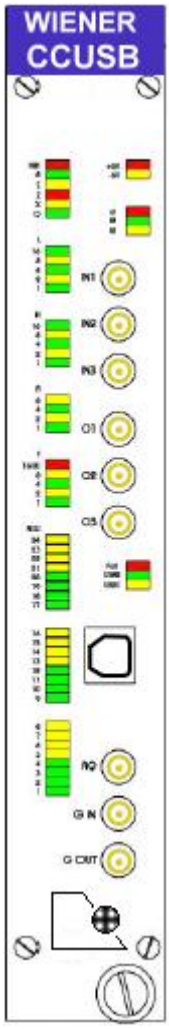
- high speed USB2 interface, auto-selecting USB2 / USB1, LED's for speed and status
- 3 pre-defined NIM, 3 user-programmable NIM (with LEMO connectors)
- 3 user-programmable LED's
- visual CAMAC data and status display with 54 red, green, and yellow LED's (N, F, A, Data, LAM, Q, X, C, Z)
- auxiliary crate controller support
- FASTCAMAC level 1 compatible
- programmable LAM mask
- direct USB-to-CAMAC calls (EASY-CAMAC)
- 1k x 16 bit memory to accommodate up to two Command Stacks for execution in autonomous list-mode operation.
- stack execution triggered either via USB link, or by a programmable combination of LAM's, or by a start signal applied to a (programmable) NIM input, or by the state of an internal timer and an event counter.
- 22-kByte pipelined data buffer (FIFO) with programmable level of transfer trigger
- Sustained readout rate in excess of 2.8 MByte/s
- Masked CAMAC command broadcast
- Low power consumption, only +6V and -6V used

1.2 Read-out/Write Modes

- single word transfer (16- or 24- bit)
- Q-stop (repeated readout of the same A and N until Q=0 is returned)
- Q-scan (repeated readout with A and N increment until Q=0 is returned)
- autonomous (intelligent) execution of commands pursuant to user-programmed stacks,
- 1k of 16-bit stack memory
- conditional command execution gated by 16-bit hit register (quadruple OR of 16-fold AND's of hit bits and programmable mask bits)
- optional (cycle-by-cycle) wait-for-LAM with programmable LAM timeout

- optional (cycle-by-cycle) skipping of S2 strobe (500ns cycles)
- stack supports Q-stop and address-scan mode entries
- stack supports FASTCAMAC mode entries
- optional readout of sub-addresses identified in a previously fetched address pattern
- block single-NAF write of up to 64 kWords (16- or 24-bit)
- block single-NAF read of up to 64 kWords (16- or 24-bit)

1.3 CC-USB Front panel



INH, B, Q, X, C, Z LED's

LAM (L1, L2, L4, L8, L16) LED's

Station (N1, N2, N4, N8, N16) LED's

Sub-Address (A1, A2, A4, A8) LED's

Function (F1, F2, F4, F8, F16) LED's

Data (9,10,11,12,13,14,15) LED's

Data (9,10,11,12,13,14,15) LED's

Data (1,2,3,4,5,6,7,8) LED's

Power LED for +6V / -6V

3 user LED's (red, green, yellow)

3 user inputs Lemo / NIM

3 user outputs Lemo / NIM

Failure LED / USB 1 or 2 indicator

USB port

Aux Controller RQ, G-in / G-out

Firmware selector (1 – 4) :
P1 – P4 for programming
C1 – C4 for use / operation

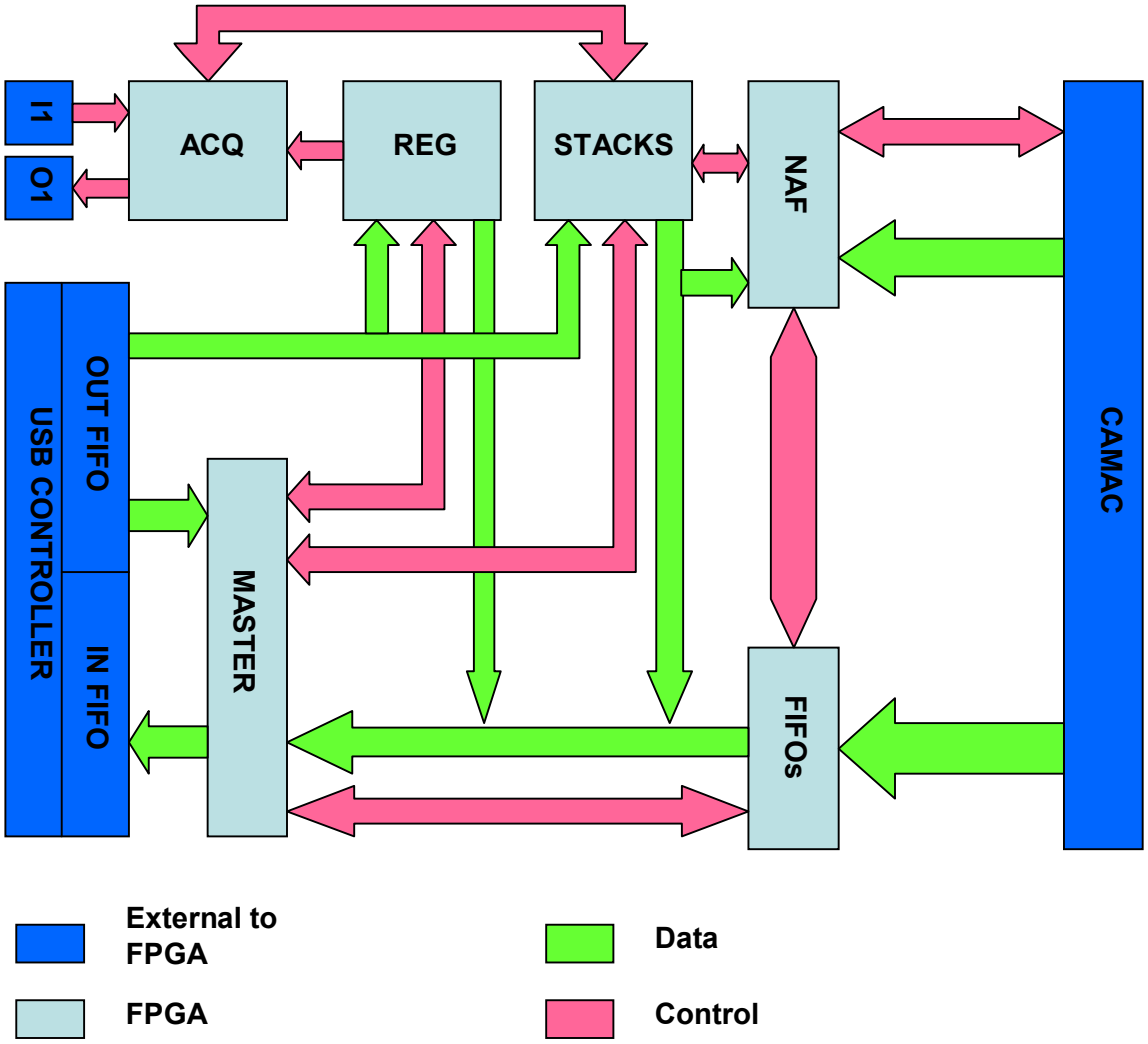
1.4 Technical Data

Packaging	double wide CAMAC module
Interface	USB2 / USB1 auto-detecting / ranging, Connector: USB type B
Inputs	3 user inputs, NIM level , LEMO type Functions pre-programmed (firmware 501): 2 x 24-bit scaler, 2 x delay gate generator inputs, DAQ trigger (I-1) coincidence register
Outputs	3 programmable outputs for CAMAC, USB and DAQ signals, NIM level, LEMO type Functions preprogrammed and selectable (firmware 501): 2 x delay gate generator / pulser outputs DAQ / control signals (busy, event trigger, event, acquire, end of busy)
Display	2 power LED's (+/-6V) 3 programmable User LED's (red, green, yellow) 3 USB status LED's (USB1, USB2, Failure) CAMAC data way display N, F, A, Data, LAM, Q, X, C, Z, I, B
Aux. Controller	Build in auxiliary crate controller support Front panel connectors for Grant In/Out and Request Rear side LAM grade connector
Firmware	Software upgradeable via USB, 4 firmware locations Selection via 8 position switch (P=program, C-use)
Performance	CAMAC up to 3MB/s, FASTCAMAC / special modes up to 12MB/s

1.5 Power Consumption

Voltage	Max. current	Power
+6 V	1.2 A	about 8 W
-6 V	0.1 A	

1.6 Block diagram



I1	- User NIM input
O1	- User NIM "Busy" output
ACQ	- Data Acquisition Control
REG	- Register Block
STACKS	- Command Stacks (2 kBytes)
NAF	- NAF Sequence Generator
CAMAC	- CAMAC Bus, Including Arbitration
FIFO's	- Three-Stage Pipe lined FIFO Array (22 kBytes)
Master	- Control Unit
USB Controller	- FX2 CY7C68013 IC
OUT FIFO	- USB Out FIFO (Relative to Host)
IN FIFO	- USB In FIFO (Relative to Host)

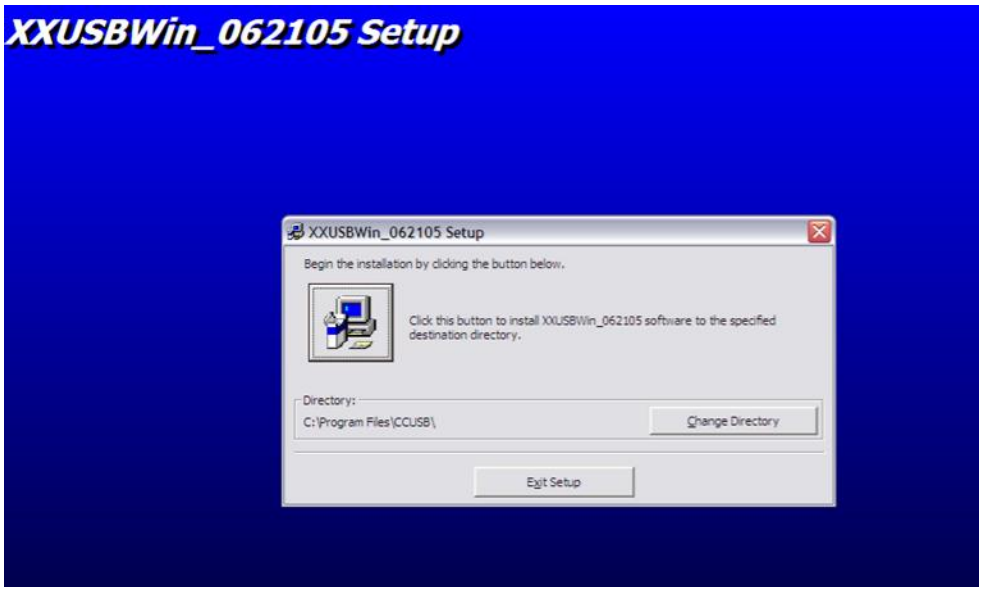
2 CC-USB AND USB DRIVER INSTALLATION

ATTENTION!!! Observe precautions for handling:

- **Electrostatic device!** Handle only at static safe work stations. Do not touch electronic components or wiring
- The CAMAC crate as well as the used PC have to be on the same electric potential. Different potentials can result in unexpected currents between the CC-USB and connected computer which can destroy the units.
- Do not plug the CC-USB into a CAMAC crate under power. **Switch off the CAMAC crate first before inserting or removing any CAMAC module!** For safety reasons the crate should be disconnected from AC mains.

2.1 Installation for Windows Operating Systems

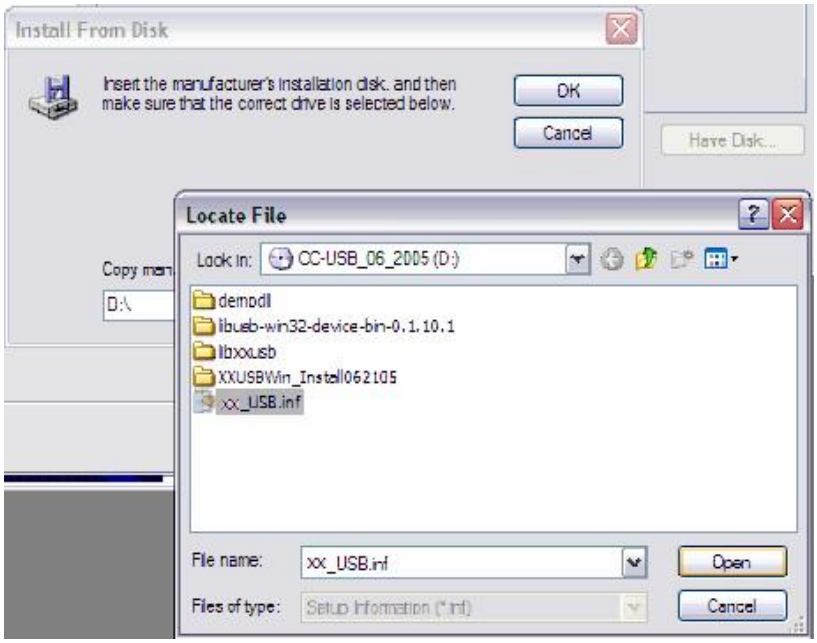
1. Switch off the CAMAC crate and remove the power cord. Plug in the CC-USB on the far right slots (normally slot 24 & 25) and secure it with the front panel screw. Switch on the CAMAC crate.
2. Insert the driver and software CD-ROM into the CD-ROM drive of the computer and run the setup program in the matching XXUSBWin_Install (98/2k/XP) folder. Define directory for installation and click the installation button.



3. Connect the CC-USB via the provided USB cable to a USB port of the computer. Running Windows 2000 or XP the hardware change should be detected and the “New Hardware Wizard” Window should open and show the CAMAC USB controller.
4. Do not use the automatic software installation but chose “installation from specific location”.

5. Select manual search for the driver
6. Type in the drive letter for the CD-ROM (e.g. D:, F:, …) and locate the file **xx_USB.inf**. Press Enter to select this driver and to close the window.

7. The WIENER CC-USB driver should be listed and highlighted in the driver list. The driver is not digitally signed which however does not have any effect on it's functionality. Press Next to finish the installation.



8. The "New Hardware Wizard" should copy all driver files into the Windows System32 folders and report a successful installation.

9. In order to use the XXUSBWin.exe program please install it by running the setup.exe program in the XXUSBWin###_Install directory (### select the right Windows version for your computer).

Please note that libusb does not allow to run multiple programs accessing the CC-USB at the same time!

2.2 Installation for Linux Operating Systems

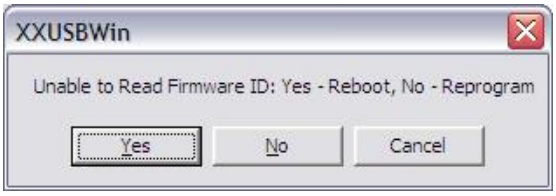
Linux support for the CC_USB is provided through a shared library and header file. To use these file simply copy them to an appropriate location, such as /usr/lib for the library and /usr/include for the header file. The functions available in the library are exactly the same as those available at for Windows and are described later in the manual. Linux specific details are located in the readme file on the software CD that you received with your module.

Libusb has permissions restrictions on driver access as well as device special file access which require root access. For non-root access permission rules have to be modified (see WIENER CC-USB forum at http://www.wiener-us.com/forums/index.php).

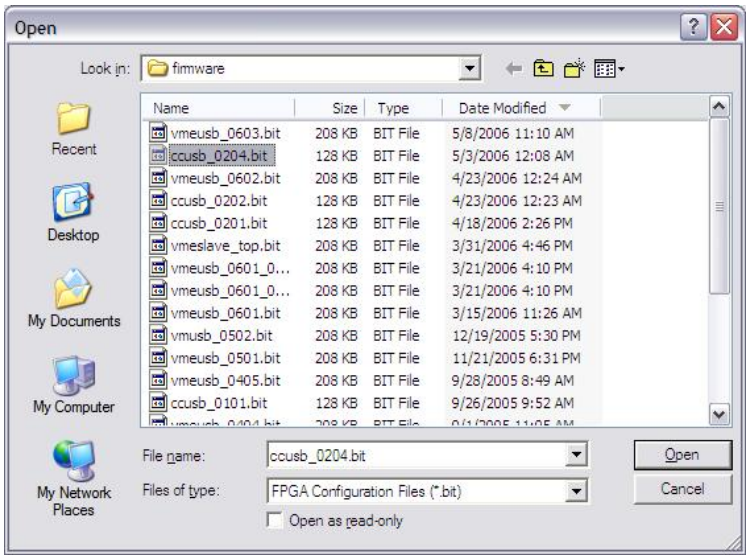
2.3 Firmware upgrades

The CC-USB is shipped with the latest firmware for the FPGA loaded however, new versions of it may be available on the web. Please occasionally check at www.wiener-d.com -> support -> downloads if newer versions of firmware, documentation and / or software are available.
The firmware upgrade is done via USB and can be performed by the help of the XXUSBWin program.

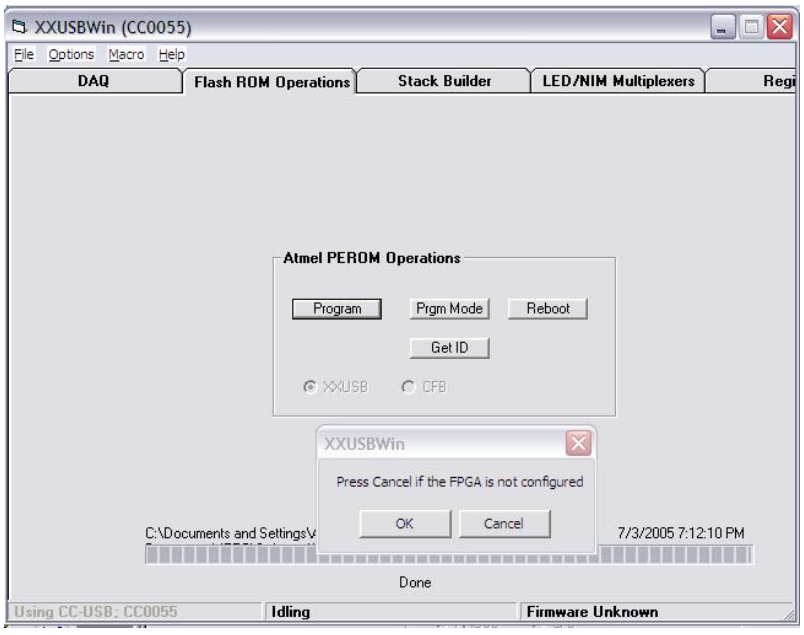
To upgrade, switch the firmware selector to one of the 4 firmware programming positions (P1-P4). The red Failure LED will be on. Start the XXUSBWin program which will show the following error message:



Select "Yes" or go to the Flash ROM Operations page and click program. Open the file of the latest firmware (xxx.bit)



When done one has to reset the controller or switch the selector switch to the corresponding run location (C1-C4) and power cycle the crate.



3 GENERAL ARCHITECTURE OF CC-USB AND ITS USER INTERFACE

The CC-USB presents to the user five internal devices or addresses shown in Table 1:
Table 1. Internal devices of CC-USB and their addresses

Address	Device
1	Register Block (RB)
2	Primary CAMAC Command Stack (PCS)
3	Auxiliary CAMAC Command Stack (ACS)
4	CAMAC NAF Generator (CNAF)
5	Common Output Buffer

3.1 Command Stacks

The CC-USB firmware allows one to set up two Command Stacks for execution in autonomous data acquisition mode. The Primary Command Stack (PCS) is intended for the sequence of commands to be executed in response to an event trigger. The Auxiliary Command Stack (ACS) is intended for a periodic execution with a user definable trigger. The ACS is intended mainly for scaler readout, but can be used also for other purposes such as, e.g., background sampling.

3.2 CC-USB Register Blocks

Since Firmware 1.01 (95001010), the Register Block "RB" of CC-USB contains only the Action Register, which needs to be accessed outside the CAMAC operations path. The remaining internal registers are accessed via CAMAC Commands with N=25as described in section 3.2.

3.2.1 Action Register (address = 1 / 0x01)

12-15	8-11	5-7	4	3	2	1	0
-	-	-	Aux Trigger	-	Clear	USB trigger	Start/stop

Bit 0 of the Action Register, when set to 1, activates data acquisition in list mode. In this mode, primary stack execution is triggered either by a start signal applied to the User NIM input I1 or a combination of LAM's coinciding with the LAM mask.

Writing "1" to Bit 1 of the Action Register generates an internal signal of 150ns duration, called USB Trigger. This signal can be routed via any of the two internal delay and gate generators to any of the three user NIM outputs O1 - O3 and/or displayed on user Green LED.

Writing "1" to Bit 2 of the Action Register clears a number of internal registers and is intended primarily for use during firmware debugging.

Writing "1" to Bit 4 of the Action Register, triggers execution of the Auxiliary Command Stack, when CC-USB is in data acquisition mode and such stack had been set up.

Bits 1, 2, and 3 are toggle bits that auto-reset in 150 ns.

3.3 Internal CAMAC Register File

The internal register file consists of 13 registers storing one constant Firmware ID and 12 words encoding static operational parameters. Additionally, there are three pseudo-registers mapping onto scaler data and the status of CAMAC LAM lines, such that the latter are accessed by issuing "Read" commands to pseudo-registers. All registers of the Internal Register File are accessed via CAMAC commands **N(25) F(0) A(i)** and **N(25)F(16)A(i)**, for "read" and "Write" access, respectively, where "i" represents the register sub-address. The functionality of the registers is shown in Table 2.

Table 2. Register sub-addresses and their functionality

CAMAC A		Register	Note
Hex	Dec		
0x0	**0**	Firmware ID	Read-Only – 32 bits
0x1	**1**	Global Mode	Read/Write – 16 bits
0x2	**2**	Delays	Read/Write – 16 bits
0x3	**3**	Scaler Readout Control	Read/Write – 24 bits
0x4	**4**	User LED Source Selector	Read/Write – 32 bits
0x5	**5**	User NIM Output Source Selector	Read/Write – 32 bits
0x6	**6**	Source Selector for User Devices	Read/Write – 32 bits
0x7	**7**	Timing for Delay & Gate Generator A	Read/Write – 32 bits
0x8	**8**	Timing for Delay & Gate Generator B	Read/Write – 32 bits
0x9	**9**	LAM Mask	Read/Write – 24-bits
0xA	**10**	CAMAC LAM (pseudo-register)	Read-Only – 24 bits
0xB	**11**	Scaler A (pseudo-register)	Read-Only – 32 bits
0xC	**12**	Scaler B (pseudo-register)	Read-Only – 32 bits
0xD	**13**	Extended Delays Register	Read/Write – 32 bits
0xE	**14**	USB Buffering Setup Register	Read/Write – 32 bits
0xF	**15**	Broadcast Map (notepad register)	Read – 24 bits

3.3.1 Firmware ID Register [Read only]

CAMAC A = 0 / 0x0

29-31	24-28	16-23	8-15	0-7
Month	Year	Beta Version	Major Revision	Minor Revision

This Firmware ID register identifies the acting FPGA firmware in eight hexadecimal digits MYBBFFRR, where M and Y represent the month and year of creation, B represents beta version, and F and R represent the firmware main and revision numbers, respectively.

3.3.2 Global Mode Register [read/write]

CAMAC A=1 / 0x1

The global mode register has the following 16-bit structure:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unused			Arbitr.	Unused			HeaderOpt	-	FrceDmp	MixBuffOpt	Unused	BuffOpt			

The BuffOpt bits (0-2) define the output buffer length. Bit 3 controls the mode of buffer filling, such that 0 closes buffers at event boundaries and 1 allows spreading events across the adjacent buffers:

BuffOpt Value	Buffer Length (words)
0	4096
1	2048
2	1024
3	512
4	256
5	128
6	64
7	Single Event

The MixBuffOpt = 1 causes regular and scaler data to share the data buffers, with scaler events identified by bit 15 set in the event header word. Conversely, MixBufOpt = 0, causes regular and scaler data to be written in separate buffers identified by bit 13 = 1 in the buffer header word. Note that by allowing scaler and regular data to share buffers, a better use is made of buffering, which results in a superior USB bandwidth, a recommended mode of operation.

The FrceDmp = 1 causes immediate dump of the data returned by the execution of the Auxiliary Command Stack.

The HeaderOpt bit controls the structure of the buffer header, such that HeaderOpt=0 writes out one header word identifying the buffer type (bit 15=1 – watchdog buffer, bit 14=0 – data buffer, bit 14=1 – scaler buffer) and the number of events in buffer. When HeaderOpt = 1, the second header word is written out listing the number of words in the buffer.

The Arbitr Bit, when set to 1 activates CAMAC bus arbitration.

3.3.3 Delays Register [Read/Write]

CAMAC A = 2 / 0x2

Bits	8-15	0-7
Function	LAM timeout	trigger delay

The delays register stores the desired trigger delay (from the start signal applied to the NIM input to the actual start of the CAMAC readout) – least significant 8 bits and the LAM timeout period – most significant 8 bits. Both delays are in units of us.

3.3.4 ACS Control Register [Read/Write]

CAMAC A = 3 / 0x3

Bits	**16-23**	**0-15**
Function	TimeInterval	NumSepEvts

The ACS Control Register defines the mode of triggering the execution of the ACS and stores the numbers defining the frequency at which ACS is to be executed during the data acquisition. There are two options for triggering the execution of ACS, with one based on the number of events collected and the other on real time lapsed from the previous execution of the ACS. Note that the execution of ACS can be also triggered by writing "1" to bit 4 of the Action Register, while CC-USB is in data acquisition mode (one must, in fact write 17 to the Action Register, to preserve the data acquisition mode). The execution of ACS is based on the "whichever-comes-first" principle, such that with more than one option active, the one that "matures" first triggers the actual readout, with all control counters being reset to zero.

NumSepEvts represents the number of data events separating the execution of ACS events. When the value is set to zero, this particular option of triggering the execution of ACS is suppressed.

TimeInterval represents the time interval in units of 0.5 s, separating the execution of ACS events. When the value is set to zero, the timer is disabled. The range of the timer is from 0.5s to 128s.

3.3.5 User LED and NIM Output Selectors [Read/Write]

CAMAC A = 4 / 0x4 - LED Register
CAMAC A = 5 / 0x5 - NIM output register

Numbers stored in these registers identify sources of User LED's and NIM Outputs. The actual selection of sources is firmware specific and subject to customization. The general bit composition of the selector word is shown in the table below

Yellow LED			**Green LED**			**Red LED**		
21	20	16-18	13	12	8-10	5	4	0-2
Latch	Invert	Code	Latch	Invert	Code	Latch	Invert	Code

NIM O3			**NIM O2**			**NIM O1**		
21	20	16-18	13	12	8-10	5	4	0-2
Latch	Invert	Code	Latch	Invert	Code	Latch	Invert	Code

The following 2-bit code identifies the source of the signal for firmware 5.01 and higher:

Code	Red LED	Green LED	Yellow LED
0	Event Trigger	Acquire	NIM I2
1	Busy	NIM I1	NIM I3
2	USB Out FIFO not empty	USB In FIFO not empty	Busy
3	USB In FIFO not full	USB Trigger	USB_In FIFO not empty

Code	NIM O1	NIM O2	NIM O3
0	Busy	Acquire	End Of Busy
1	Event Trigger	Event	Busy
2	DGG_A	DGG_A	DGG_A
3	DGG_B	DGG_B	DGG_B

Note 1. “Busy” signal indicates that stack processing is in progress, with CAMAC operations not having completed. “Busy” is asserted when event readout is triggered and de-asserted as soon as CAMACCAMAC operations are completed.

Note 2. “Acquire” indicates that the data acquisition mode is active.

Note 3. “USB Trigger” is generated in response to writing to bit 1 of Action Register.

Note 4. “Event Trigger” indicates that event readout has been triggered.

Note 5. Invert bit causes the signal to be inverted

Note 6. Latch bit causes the signal to be latched. To release the latch one must toggle the bit.

Note 7. DGG_A and DGG_B are output pulses of the two user delay and gate generators.

3.3.6 User Devices Source Selector [Read/Write]

CAMAC A = 6 / 0x6

In addition to the two NIM outputs, firmware 95000101 implements four user devices - two delay and gate generators and two 32-bit scalers. All of these devices may use various signals as input/trigger signals, with the selection identified by respective code bits stored in the User Devices Source Selector register. Additionally, this register accommodates six scaler operation bits that allow one to enable/disable and clear the two scalers. The bit composition of the User Devices source selector register is shown in the table below.

Device	Freeze Reg.	Reset	Enable	Code
SCLR_A	6	5	4	0-2
SCLR_B	14	13	12	8-10
DGG_A		-	-	16-18
DGG_B		-	-	24-26

Writing to scaler operation bits is intrinsically separated from writing to code bits, such that the content of the latter is preserved during scaler enable/disable and reset operations. Note that scaler disabling requires writing of zero into the Enable bit and must hence be identified additionally by writing 1 to the Freeze Reg. bit.

To enable or reset a scaler one must write "1" into the respective Enable or Reset bit. To disable a scaler, one must write simultaneously "1" into the respective Freeze Reg. bit and "0" into the respective Enable bit. All enable/disable and reset operations may be performed in one "write" operation into the Source Selector Register, by setting the desired values of all six control bits.

The Scaler and DGG input selector codes are defined as:

Code	SCLR_A	SCLR_B	DGG_A	DGG_B
0	Disabled	Disabled	Disabled	Disabled
1	NIM I1	NIM I1	NIM I1	NIM I1
2	NIM I2	NIM I2	NIM I2	NIM I2
3	NIM I3	NIM I3	NIM I3	NIM I#
4	Event	Event	Event Trigger	Event Trigger
5	Term SCLR_B	Term. SCLR_A	End of Event	End of Event
6	DGG_A	DGG_A	USB Trigger	USB Trigger
7	DGG_B	DGG_B	Pulser	Pulser

Note: Term SCLR_A and Term SCLR_B are terminal counts of the two scalers. They allow one to daisy-chain the two 24-bit scalers into one 48-bit one.

To make use of any one of the two scalers SCLR_A and SCLRB one must first define the input signal by writing a proper code into the designated field of the Source Selector Register. Then by setting the "freeze bit" one can perform the scaler operations (reset, enable) without disturbing the settings of the input selector codes stored in this register.

Note, that one can daisy-chain the two scalers into one 48-bit scaler by selecting the terminal count of one scaler as an input signal to the second scaler. This can be used as long time stamps by selecting the output signal of a DGG configured as a pulser as an input signal for the first scaler of the daisy-chain.

3.3.7 Delay and Gate Generator Registers DGG_A and DGG_B [Read/Write]

CAMAC A = 7 / 0x7 - Delay and Gate Generator A
CAMAC A= 8 / 0x8 - Delay and Gate Generator B
CAMAC A= 13 / 0xD - Extended Delays (coarse)

The two Delay and Gate Generator Registers DGG_A and DGG_B as well as the DGG/P_A/B Extend register store data defining the length of the delay and the length of the gate in units of 10 ns (100 MHz clock) for either the gate and delay generator or for the pulser. These values can be set for channel A and B independently. The pulser is re-triggering after the defined delay time, i.e. the delay time + gate length defines the pulser repetition rate. With firmware revision 5.01 both DGG's have a 24-bit range for the delay setting. The value of the delay is a composite of a 16-bit high resolution value (10ns) and an 8-bit coarse range value. Earlier firmware versions use only the fine (10ns) value.

Gate length = 10ns * Gate ***Delay = 10ns * Delay_fine + 655.36µs* Delay_coarse*** ***Pulser repetition period= Gate + Delay***

DGG_A (A=7)

Bits	DGG_A 16-31	DGG_A 0-15
Function	Gate	Delay_fine

DGG_ B (A=8)

Bits	DGG_B 16-31	DGG_ B 0-15
Function	Gate	Delay_fine

DGG_ Ext (A=13)

Bits	DGG_B Ext (16-31)	DGG_A Ext 0-15
Function	Delay coarse (8bit)	Delay coarse (8 or 16 bit)

3.3.8 Scaler Registers SLR_A and SCLR_B [Read only]

CAMAC A =11 / 0xB - Scaler A
CAMAC A = 12 / 0xC - Scaler B

Scaler registers store 24-bit scaler data in a straightforward manner. The use of the scalers is described further below. Some firmware versions before 4.02 as well as 5.02 were/ are available with 32-bit scalers.

3.3.9 LAM Mask Register [Read/Write]

CAMAC A = 9 / 0x9

The LAM Mask Register is a 24-bit register that stores the LAM Mask defining what combination of LAM's triggers event readout during the data acquisition. When zero, the readout is triggered by a signal applied to the NIM input. The LAM mask is only used for DAQ mode triggering and will not effect the LAM display or other LAM functions.

3.3.10 USB Bulk Transfer Setup Register [Read/Write]

CAMAC A = 14/0xE

To benefit from the high bandwidth of the USB2 interface, one needs to avoid overheads associated with any single transfer operation. Therefore, one must strive to reduce the number of transfers by extending the length of bulk transfers. CC-USB, by default closes USB buffer (generates a "packet end") either at the end of the data buffer or at the end of event. This guarantees bulk transfer lengths of only 8 kBytes for short events and lengths equal to event lengths, in the case of long events. Such default setting does not allow one to utilize the USB2 bandwidth when short events are acquired and, therefore, CC-USB offers an option to "bundle" multiple data buffers together for a single bulk USB transfer. Since in the case of short events

the time of filling multiple buffers is variable, the option includes setting of a watchdog timer, which will guarantee that a "packet end" signal is generated at timeout, should the data buffers fail to fill sufficiently fast. This watchdog timeout should be made shorter than the software timeout set for bulk read. The relevant numbers for multi-buffer bulk transfer are stored in the USB Bulk Transfer Setup Register at A = 14 (0xE) such that the number of buffers is specified in bits 0 – 7 of this register and the timeout is specified in bits 8-11. The 4-bit timeout represents the number of seconds in excess of 1s, after which the packet end signal is issued, should the specified number of buffers not be completed by that time. Note that the default (minimum) is 1s.

Bits	12-31	8-11	4-7	0-3
Value	-	Time out	Number of buffers	

3.3.11 Broadcast MapRegister [Write-Only]

CAMAC N=27, A=D(0-3), F(4)=1, F(0-3)=D(4-7)

CC-USB allows to broadcast Write and Control commands to a number of selected modules identified by bits set in the Broadcast Map Register. Because of the internal CC-USB architecture writing to this register is done in a byte-serial fashion where consecutive bytes of the 24-bit registers are written using CAMAC A(0-3) and CAMAC F(0-3) to encode the data. To write a byte, one issues a CAMAC command with N=27, A equal to low nibble (4 bits) of the data byte, and F(0-3) equal to high nibble of the data byte. F(4) must be set to 1 as for a "write" command. Consecutive commands fill the consecutive bytes of the register. Any command other than a Broadcast Map Register "write" command resets the byte counter to zero.

3.3.12 Broadcast Map Notepad Register [Read]

CAMAC N=25, A=15, F=0

The Broadcast Map Notepad Register contains a copy of the Broadcast Map Register (BMR) to allow one to verify what was written into the BMR.

3.4 CAMAC NAF Generator / EASY-CAMAC

The CC-USB allows a direct access from the computer via USB to the CAMAC bus and modules, which is called "Easy-CAMAC". These calls can be either simple CAMAC commands or be more complex to allow special CAMAC modes as Q-stop, Q-scan, … (see detailed description in chapter 4.4 an 4.5). Due to the USB latency time the EASY-CAMAC calls are limited to a few kHz rate. EASY-CAMAC commands can be performed with the provided CCUSB-WIN program Stack-builder window). For user programs a library of standard CAMAC calls is part of thelibxxusb.dll for MS Windows or libxx_usb.so for Linux operating systems.

3.5 CAMAC common functions

The common CAMAC controller functions as Initialize (Z), Clear (C) and Inhibit (I) are realized via NAF calls to "internal" station numbers N=28 and 29. These functions can be programmed as follows

Function	N	A	F
Z	28	8	29
C	28	9	29
Set Inhibit	29	9	24
Clear Inhibit	29	9	26

3.6 Broadcast Write and Control Commands

CC-USB allows one to execute broadcast write and control commands to modules identified by bits in the Broadcast Map Register. Broadcast commands are generated by setting N=26 and using the desired A, F, and D.

3.7 Writing a Marker Word into the Output Data Stream

Firmware *0301 and newer offers the capability to insert into the output data stream marker words to facilitate viewing and interpreting the content of data buffers. A typical use of such marker words is to mark the end of an event or end of long blocks of data. Note that firmware *0301 and newer no longer marks end of events by 0xFFFF, leaving the user the flexibility of deciding on the number and the appearance of end of event markers.

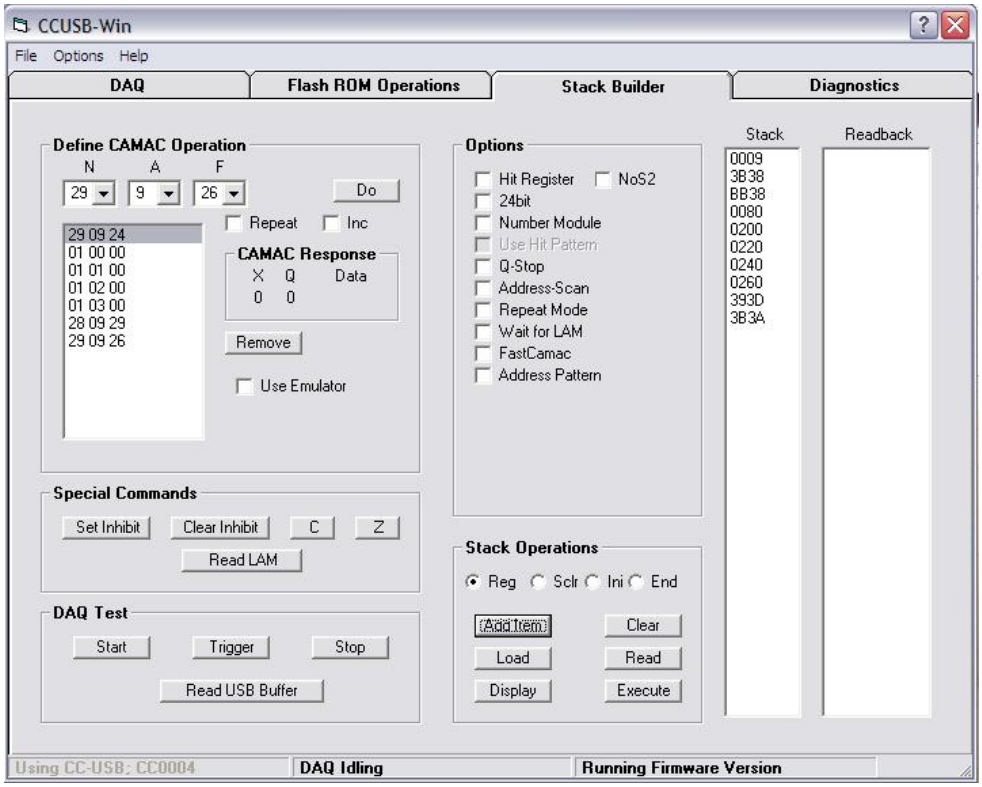
3.8 Command Stacks

For maximum performance all CAMAC commands have to be stored and executed from the CC-USB Command Stack. The stack with 1k x 16bit size can consist of up to 1000 simple CAMAC operations but also consist of more complex operations. The following commands can be programmed (others may be added in future firmware versions):

- CAMAC NAF
- CAMAC NAF read 16bit / 24bit
- CAMAC NAF write 16bit / 24bit
- C, Z, I (as NAF, see paragraph 4.6)
- LAM mode
- Hit data / hit mode,
- Q-stop
- Address scan
- Repeat mode, number data
- Fast CAMAC L1
- Broadcast write and control commands

3.9 Using the XXUSBWin Application

The MS Windows application XXUSBWin allows to create, save and read as well as to upload the Command Stack list in an easy and convenient way. Further it is possible to create the Command Stack with either a text editor or user program. All required programming details are given in chapter 4.5.



3.10 CC-USB CAMAC Function Table

N	A	F	Function	Data
0	*	16	Write a 16-bit marker word into the output data stream	16
1…24	*	*	Executes N(1..24) A(*) F(*) command on CAMAC data way	16/24
25	0	0	Read Firmware ID	32
25	1	0	Read Global Mode	16
25	1	16	Write Global Mode	16
25	2	0	Read Delays	16
25	2	16	Set Delays	16
25	3	0	Read Scaler Readout Control	24
25	3	16	Write Scaler Readout Control	24
25	4	0	Read User LED Source Selector	32
25	4	16	Write User LED Source Selector	32
25	5	0	Read User NIM Output Source Selector	32
25	5	16	Write User NIM Output Source Selector	32
25	6	0	Read Source Selector for User Devices	32
25	6	16	Write Source Selector for User Devices	32
25	7	0	Read Timing for Delay & Gate Generator A	32
25	7	16	Write Timing for Delay & Gate Generator A	32
25	8	0	Read Timing for Delay & Gate Generator B	32
25	8	16	Write Timing for Delay & Gate Generator B	32
25	9	0	Read LAM Mask	24
25	9	16	Write LAM Mask	32
25	10	0	Read CAMAC LAM (pseudo-register)	24
25	11	0	Read Scaler A	24
25	12	0	Read Scaler B	24
25	13	0	Read Extended Delays Register	8
25	13	16	Write Extended Delays Register	8
25	14	0	Read USB Buffering Setup Register	32
25	14	16	Write USB Buffering Setup Register	32
25	15	0	Read Broadcast Map (notepad register)	24
26	*	*	execute Broadcast A(*) F(*) on CAMAC dataway	16/24
27	**	**	Set Broad cast mask (3 sequential calls for 24 bit mask)	24
28	8	29	CAMAC Z	-
28	9	29	CAMAC C	-
29	9	24	Set CAMAC I	-
29	9	26	Clear CAMAC I	-

4 COMMUNICATING WITH CC-USB

Communication with the CC-USB consists in writing and reading of buffers of data to/from the USB2 port of the CC-USB using bulk-transfer mode. Borrowing from the USB language, the buffers to be written to the CC-USB will be called Out Packets, and they are sent to pipe 0 of the USB port. The buffers to be read will be called In Packets, and they are read from pipe 2 of the USB port.

The USB controller IC, when connected to a USB2 port configures packet lengths to 512 bytes. For USB1 (full speed), the packet length is set to 64 bytes. The Out Packets must be properly formatted to be understood by the internal devices of CC-USB and, by the same token, the format of the In Packets retrieved from the CC-USB must be understood by the user in order to be useful.

User may send Out Packets to four devices – the Register Block (RB), CAMAC Readout Stacks (CDS and CSS), and the NAF Generator (RB, CDS, CCS, CNAF). User may read In Packets only from the Common Output Buffer. Reading back data from the RB, CDS, and CSS is achieved by, first sending a data request Out Packet to these devices and then by reading the In Packet containing the requested data from the Common Output Buffer.

Writing to the CAMAC NAF Generator constitutes implicitly a request for data, such that in response to such a writing, CC-USB performs the requested CAMAC operation and returns the CAMAC data in the Common Output Buffer. Both, In and Out Packets are of a variable length, depending on which internal address is involved and what the content of the message is.

Important Note:

With some drivers (EZUSB in conjunction with Windows API), read operations from the USB port are blocking operations such that the host program will stop executing until the data is available at the port. Therefore, the host program must make sure (by first requesting data) that the CC-USB has placed data in the Common Output, before the read command is issued. The CC-USB provides a mechanism for supplying data, even when the host program is “frozen” in a state of waiting for data. The mechanism consists in starting a second copy of the program and issuing a bare request for data command from this second copy, not followed by the read IN Packet command.

The libxxusb package of CC-USB access functions makes overlapped USB calls that have preset timeout periods. When no data is available until the end of this period, the I/O is canceled and the respective function returns error code. The user is then expected to take proper actions, which may include resubmitting the call.

It is important to specify a sufficiently long In Packet size to be at least of the size of the actual data buffer available at the Common Output Buffer. This is especially important in the case of reading CAMAC data buffers, which differ in size substantially depending on the structure of the CAMAC Readout Stack.

4.1 General structure of Out Packets

Since internally, the USB controller of the CC-USB is set up as a 16-bit wide FIFO (First- In-First-Out Memory), the In and Out Packets are organized as collections of 16-bit words. For

the purpose of the software, and more specifically, of the Windows Application Programming Interface (API) routines, the data are packed in byte-wide buffers, a process that may remain transparent to the user when proper sets of routines (DLLs) are used. Also, much of the technical information on writing and reading back data from the internal devices of the CC-USB may be considered redundant, when a set of routines is available to perform the task. This information is, however, necessary for writing such routines.

First (16-bit) word in an Out Packet identifies the internal device/address for which the packet is intended and whether the packet represents a request for data or represents the data to be stored/interpreted to/by the target device. The latter information is coded in bit 3 (value=4) of the header word, with bit 3 set to write data. The meaning of the second word in the Out Packet depends on the address and represents the sub-address in the case of the Register Block and the number of words to follow, in the case of the CAMAC Stacks (CDS and CSS) and the CAMAC NAF Generator (CNAF). The subsequent words in the buffer, if any, represent the data to be stored in the target device or the data to be interpreted and acted upon by the target device (in the case of the CNAF). A detailed description of Out Packets for the four target devices is given below.

4.2 Writing Data to the Register Block

The Out Packet for writing data to the register block is used only for the action register (see 3.1). All other registers are accessed via the CAMAC command generator. The Out Packet is composed of the following words:

1. Target Address + 4 = 5 the target address of the register Block + the write flag (bit 3)

2. Register Sub-Address sub-address of a particular register in the block (see Table 2, further above).

3. Data To be Written a 16-bit data word.

4.3 Reading Back Data from the Register Block

To read back data from the Register block, one must first send a request Out Packet to the Register Block consisting of two words:

1. Target Address = 1 the target address identifying the register block

2. Register Sub-Address sub-address of the register of interest (see Table 2.)

4.4 Writing Data to the Command Stacks and to the NAF Generator

The Out Packets targeting the two CAMAC Stacks and the CAMAC NAF Generator (EASY-CAMAC) have identical structure, differing only in the Target Address and in length:

1. Target Address 2, 3, or 8, for CDS, CSS, and CNAF, respectively

2. Number of words in the stack (N)

3. N stack words

The Primary CAMAC Command Stack (Address=2) is intended for storing information on the sequence of the CAMAC commands to be performed when an event trigger is detected. The Auxiliary CAMAC Command Stack (Address=3) is intended mainly for storing information on the sequence of the scaler readout commands, when a periodic readout of scalers is desired. The CAMAC NAF Generator (Address 8) is an internal module that interprets the information found either in the CAMAC Stacks (when CC-USB is in data acquiring mode) or in the Out Packet received from the USB port (when CC-USB is in interactive mode).

4.5 Structure of the CAMAC Stack

A CAMAC stack consists of a sequence of properly encoded simple (one line for CAMAC "Read" commands and 3 lines for CAMAC "Write" commands) or complex (multi-line) CAMAC commands.

Simple commands specify only the desired N, A, and F to be issued by CC-USB and, additionally, whether the data is 24-bits (Long Mode) or 16-bits long For the CAMAC "Write" commands, additional one or two lines specify the data to be written depending on the state of the "Long Mode" bit in the figure below.

The data word for a simple command has the following structure:

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	**0**	**Long Mode**	**N**					**A**				**F**				

This data word can thus be calculated as

Command = F+32*A+512*N+16384*LongMode

CAMAC "Read" operations have F values from 0 through 7. CAMAC "Write" operations have F values from 16 through 23. All other F values are non data transfer (control) operations and should be operated with a "Read" call when using the XXUSB library.

Complex commands are possible for Read and Write operations. The first word of a complex commands is similar to a simple command, except that it has the continuation bit (bit 15) set:

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	**1**	**Long Mode**	**N**					**A**				**F**				

The second word is an options word, detailing the mode of readout to be performed or the nature of the data to be read. Depending on which bits in the second word are set, a number of additional words, if any, will follow. The continuation bit **C** (bit 15) of the second word is set whenever additional data are to follow. The structure of the options word is as follows:

Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	**C**	**-**	**NT**		**-**	**X**	**AP**	**FC**	**LM**	**RM**	**AS**	**QS**	**HM**	**ND**	**S2**	**HD**

Where the individual bits have the following meaning:

HD Hit Data - identifies the data as a 16-bit hit register data (coincidence register data), to be used for the conditional readout of subsequent CAMAC modules

S2 When set, S2 strobe is suppressed, the CAMAC cycle ending at the end of S1

ND Numbers Data - identifies the data as representing the number of times the next command in stack has to be performed.

HM Hit Mode - instructs the NAF Generator to condition the readout with the content of the hit pattern read in the first command of the stack (first command in an event). The Number of Product Terms used to condition the readout must be specified as well.

QS Q-stop mode – the command is to be repeated as long as Q=1 (Q response from the addressed CAMAC module), but not more than the number specified in the following stack line..

AS Address Scan – the command is to be repeated a number of times specified in the following word of the stack, with A incremented by 1 each time.

RM Repeat Mode – repeat command a number of times specified in the following stack line.

LM LAM Mode – wait for LAM from station N specified in first word, subject to LAM Timeout and perform the readout only when LAM is set (Not to be used with xxusb_stack_execute calls!).

FC Fast CAMAC Mode – perform the readout in Fast CAMAC mode a number of times specified in the following stack line.

AP Address Pattern Data – identifies the data as an address pattern to be used in conjunction with the command that follows. The subsequent command will be repeated for every address for which the bit is set in the address pattern data word.

X When set to 1 Q and X response to write commands is recorded in the data buffer.

NT Number of Product Terms – specifies the number of words in the stack that follow and that constitute bit masks for constructing a logical equation used in deciding whether the given operation is to be performed for the particular hit register data.

C Continuation bit, set to 1 in case additional words are following

The following rules apply:

(i) Whenever the Repeat Mode (RM), Address Scan (AS), Q-Stop, or FC bit is set, the stack line must be followed by another line defining the maximum number (up to 0xFFFC= 65532) of times the command is to be repeated. Note that the large numbers of repetitions are suitable only for single-NAF commands.

(ii) Whenever a write command is issued in conjunction with the Repeat Mode (RM) (single NAF block write), the first data word is to follow the first command line and the remaining words follow the Number of Repetitions word. The number of repetitions may be as high as 0xFFFC=65532.

(iii) When the Hit Mode (HM) bit is set, the Number of Terms bits must be declared. The stack line must be followed by the specified number of data lines representing bit masks BMask(1 to NT), to be used in constructing the logical condition for performing the command. The logical equation is:

[BMask(1) AND HD = BMask(1)) OR (BMask(2)
AND HD = BMask(2)) OR (BMask(3)
AND HD = BMask(3)) OR (BMask(4)
AND HD = BMask(4)],

i.e., the command will be performed whenever all bits in any of the specified Bit Masks are set in the hit register data.

Since the stack can be quite complex, it is advisable to write a proper routine to set up the stack. As an option, one may utilize the XXUSBWin Windows application to build the stack and save it to disk.

The following example shows the Command Stack (as saved to file) for a simple 4-parameter readout. Explanations are added in blue color:

```
CCUSB CAMAC Stack Generated on 8/10/2006 at 1:21:01 PM
 7          // number of lines
0200        // read 1. channel  N(1), A(0), F0)
0220        // read 2. channel  N(1), A(1), F0)
0240        // read 3. channel  N(1), A(2), F0)
0260        // read 4. channel  N(1), A(3), F0)
393D        // clear C - N(28), A(9), F29)
0010        // write end marker N(0), A(0), F16)
FFFF        // end marker 0xffff
```

4.6 Structure of the IN Packets

The General Output Buffer is associated with Endpoint 6 of the USB2 controller IC, which is configured as a 512 byte deep FIFO. This endpoint is configured for bulk transfer and one can specify lengths of buffers to be read of any length (up to 8192 bytes) compatible with the CC-

USB functionality. Al data supplied by the CC-USB is to be read from the Endpoint 6. While reading, it is important to specify the length of the buffer not shorter than the length of the actual data buffer written by the CC-USB into this endpoint.

The structure of data retrieved in conjunction with direct requests for data addressed to the Register Block and to the CAMAC Stacks is simple, such that the buffer consists only of the requested data. For write commands CC-USB returns only XQ in one 16-bit word when the write command is the last command in the stack – to guarantee that there is always at least one word returned as an acknowledgment of a stack execution.

The data buffers read during the data acquisition process have a structure depending on the mode of buffering, i.e., whether event data are allowed to span two buffers (bit 3 of BuffOpt set). Additionally, there are special rules for treating long events. These are discussed at the end of this section.

For the Integer Event Mode, the data buffer has the following structure:

1. Header word Bit 15=1 indicates a watchdog buffer, bit14=1 indicates a scaler buffer. Bits 0 – 9 represent the number of events in the buffer.

2. Optional 2nd Header Word Bits 0-11 represent the number of words in the buffer.

3. Event Length Event length including terminator words.

4-N1. Event Data

N2. Event Terminator 0xFFFF not applicable to firmware *0301 and higher

N3. Optional 2nd Terminator 0x FFFF not applicable to firmware *0301 and newer

...

. Subsequent Events

...

N5. Buffer Terminator hex FFFF

The unpacking of the events must be done in accordance with the CAMAC Stack that is involved in generating the buffer.

In the Split-Event mode, when events span two or more buffers, no buffer terminator is written.

For the direct access of the CAMAC NAF generator (EASY-CAMAC), no header words are written and the “In Packet” contains only one event with data and or Q / X.

CAMAC Write: returns 1 word with Q, X:

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	Q

CAMAC 16-bit Read: returns 1 word with data 0-15:

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1.	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

CAMAC 24-bit Read: returns 2 words, with data 0-15 / data 16-23 and Q/X:

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1.	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
2.							X	Q	D23	D22	D21	D20	D19	D18	D17	D16

CC-USB has dedicated 2kWords-long event FIFO to assemble events. To handle longer events, CC-USB splits the long event into parts, each of which appears as a separate event in the output buffer. The partial events are distinguishable by bit 12 of the Event Length word set, except for the last part. Also, only the last “installment” is terminated by the Event Terminator word (s).

CC-USB has a provision to automatically change the output buffer packing mode to Split-Event mode, whenever the Event Length exceeds the length of the Integer-Event buffer. The occurrence of such a change is indicated by setting of bit 13 in the buffer header word.

CAMAC calls of the Control group with 7<F<15 and F>23 are performed like CAMAC_Read calls but will return Q and X response similar to a write call.

5 GUIDE TO LIST MODE DATA ACQUISITION WITH CC-USB

CC-USB is intended for use in list mode data acquisition, where it performs sequences of CAMAC commands defined in stack(s) upon receipt of event trigger. CC-USB then formats the data read form the CAMAC bus and buffers them in a data buffer. When the buffer is full the content is transferred to the In-FIFO of the USB controller IC for readout by host software.
To set up CC-USB for data acquisition in list mode the following steps are recommended:

1. Build the Primary and/or Auxiliary CAMAC Command Stacks by adding all the desired simple and complex commands to it. One must make sure that the stack sequence will clear all CAMAC modules that need to be cleared. It is recommended to first execute the stack from the host software to verify that it performs as intended. For this purpose the libxxusb library function xxusb_stack_execute can be used.
2. Load the stack(s) into the CC-USB memory using the libxxusb library function xxusb_stack_write. It is recommended to read back the stack (function xxusb_stack_read), to verify that the stack is correctly stored.
3. Set up the data acquisition trigger mode. By default, CC-USB commences execution of the stack upon receipt of a NIM signal at its user NIM input I1.
4. Set the trigger delay (time from the receipt of an event to the commencement of the stack execution) and LAM timeout.
5. Set up buffering mode and data buffer length by writing a suitable 4-bit code into bits 0-3 of the Global Mode Register. The default is buffer length of 4096 words and events fitting into one buffer.
6. Set up CAMAC bus arbitration, if necessary. The default is no arbitration.
7. Set buffer header option. By default, CC-USB writes one buffer header word containing information on the number of events in the buffer, buffer type (regular, or periodic scaler), and the buffer termination mode (regular or watchdog).
8. Start acquisition by setting bit 0 of the Action Register to 1. End acquisition by resetting this bit to “0”. While in acquisition mode, the host software is expected to read the USB port In FIFO in a loop, to empty it and make space for subsequent events.
9. After ending data acquisition all buffers should be emptied by continuing reading data until no data are available anymore.

6 LIBXXUSB LIBRARY FOR WINDOWS AND LINUX

A dedicated library of functions was developed to facilitate the utilization of CC-USB and its VME counterpart, VM-USB. This library libxxusb requires the libusb0.sys driver to be installed. It is in fact a wrapper library for the general-use open-source libusb-win32 library available via www.sourceforge.net. All functions are part of the libxxusb.dll / libxxusb.so dynamically loadable libraries.

For linux the library is called libxx_usb. All the functions are identical to the ones used in Windows.

All xxusb functions for both 32-bit MS Windows (Win98SE, WinME, Win2k, WinXP) as well as for Linux rely on the USB library "libusb-win32"(Windows) or "libusb" (Linux). For further details about these libraries please see www.sourceforge.net or http://sf.net/projects/libusb/ .

The following functions are used for both the CC_SUB and its' counterpart the VM_USB.

6.1 xxusb_devices_find

The xxusb_devices_find function retrieves relevant parameters of USB ports of all XX-USB devices attached to the host and returns these in an array of proper structures. This is the first command to be issued when attempting to establish communication with an XX-USB.

```
WORD xxusb_devices_find{
    XXUSB_DEVICE_TYPE lpXXUSBDevice,
};
```

Parameters

lpXXUSBDevice

[out] Pointer to an array of structures storing parameters of all XX-USB devices identified.

Return Values

On success, the function returns the number of XX-USB devices found, including 0.
A negative return value indicates that the handle to a valid device could not be opened as a result of insufficient privileges. It is recommended to retry in Superuser mode.

6.2 xxusb_device_open

The xxusb_device_open function obtains handle to the desired XX-USB device, identified by xxusb_devices_find command. This is the second command to be issued when attempting to establish communication with an XX-USB. The obtained handle is then to be used while calling various xxusb_*_* functions, that require the handle. Upon termination of a XX-USB session, the handle is to be released by calling xxusb_handle_close.

WORD xxusb_device_open{
USB_DEVICE_TYPE lpUSBDevice,
};

Parameters
lpUSBDevice
[in] Pointer to a structure storing parameters of the target XX-USB devices.

Return Values
On success, the function returns the handle to the target XX-USB device. A negative return value indicates that the handle to a valid device could not be opened as a result of insufficient privileges. It is recommended to retry in Superuser mode.

Remarks
While all xxusb functions rely on the libusb (www.sourceforge.net) functions while communicating with XX-USB, xxusb_device_open and xxusb_handle_close are simply macros creating aliases to usb_open and usb_close functions of the libusb library.

6.3 xxusb_serial_open

Opens a xxusb device (CC-USB or VM-USB) whose serial number is given

USB_DEV_HANDLE* xxusb_serial_open{
char *SerialString
};

Parameters:
SerialString
a char string that gives the serial number of the device you wish to open. It takes the form:
VM0009 - for a VM_USB with serial number 9 or
CC0009 - for a CC_USB with serial number 9

Returns:
LpUSBDevice
[out] Pointer to a variable containing the handle to the controller

6.4 xxusb_device_close

The xxusb_device_close function closes the handle to the desired XX-USB device, obtained by a xxusb_device_open call. This function is to be called upon termination of an XX-USB session.

WORD xxusb_device_close{
USB_DEV_HANDLE lpUSBDevice,
};

Parameters
lpUSBDevice
[in] Pointer to a variable containing the handle to be closed.

Return Values
Returns negative upon failure.

Remarks
While all xxusb functions rely on the libusb (www.sourceforge.net) functions while communicating with XX-USB, xxusb_device_open and xxusb_handle_close are simply macros creating aliases to usb_open and usb_close functions of the libusb library.

6.5 xxusb_reset_toggle

The xxusb_reset_toggle function toggles the reset state of the FPGA while XX-USB is in programming mode – rotary selector set in one of four P* positions.

WORD xxusb_reset_toggle{
HANDLE hDevice,
};

Parameters
hDevice
[in] Handle to the XX-USB device.

Return Values
Returns negative upon failure.

6.6 xxusb_register_write

The xxusb_register_write sends a data buffer to XX-USB, causing the latter to store the desired data in the target register.

WORD xxusb_register_write{
HANDLE hDevice,
WORD wRegisterAddress,
DWORD dwRegisterData
};

Parameters
hDevice

[in] Handle to the XX-USB device.

wRegisterAddress
[in] Address of the XX-USB register.

CC-USB has only one register which is the Action Register at address 1.

dwRegisterData
[in] Data to be stored in the register.

Return Values
On success, the function returns the number of bytes sent to XX-USB.
Function returns 0 on attempted writes to read-only registers and negative numbers on failures.

6.7 xxusb_register_read

The xxusb_register_read function first, sends a buffer to XX-USB, causing the latter to write the content of a desired register to its USB port FIFO and then, obtains the value by reading the buffer from the XX-USB.

```
WORD xxusb_register_read{
    HANDLE hDevice,
    WORD wRegisterAddress,
    LPDWORD lpRegisterData
};
```

Parameters
hDevice
[in] Handle to the XX-USB device.

wRegisterAddress
[in] Address of the XX-USB register.

CC-USB has only one register which is the Action Register at address 1.

lpRegisterData
[out] Pointer to a variable that receives the data returned by the operation, i.e., the value stored at wRegisterAddress of XX-USB.

Return Values
On success, the function returns the number of bytes read XX-USB. Valid value is 2.
Function returns a negative number on a failure.

6.8 xxusb_stack_write

The xxusb_stack_write function sends a buffer to XX-USB, causing the latter to store this content in a dedicated block RAM, for use when data acquisition mode is active. This content can be read back using *xxusb_stack_read* function.

WORD xxusb_stack_write{
HANDLE hDevice,
WORD wStackType,
LPDWORD lpStackData
};

Parameters
hDevice
[in] Handle to the XX-USB device.

wStackAddress
[in] Type of the XX-USB stack, the content of which is to be read. Valid types are 2, for the regular stack and 3, for the periodic (scaler) readout stack.

lpStackData
[in] Pointer to a variable array that contains the data to be stored in the target stack.

Return Values
On success, the function returns the number of bytes sent to XX-USB. The latter value is twice the length of the stack plus 2 (for a header word identifying a stack as a target).
Function returns a negative number on a failure.

Remarks
The physical length of the regular stack is 768 16-bit words for CC-USB and 768 32-bit words for VM-USB.
The physical length of the periodic (scaler) stack is 256 16-bit words for CC-USB and 256 32-bit words for VM-USB.
While the stack is expected to contain properly encoded sequence of CAMAC (CC-USB) or VME (VM-USB) commands to be performed by XX-USB, it can store any sequence of numbers.

6.9 xxusb_stack_read

The xxusb_stack_read function first, sends a buffer to XX-USB, causing the latter to write the content of a desired stack to its USB port FIFO and then, obtains this content by reading a buffer from the XX-USB.

WORD xxusb_stack_read{
HANDLE hDevice,
WORD wStackType,

LPDWORD lpStackData
};

Parameters
hDevice
[in] Handle to the XX-USB device.

wStackAddress
[in] Type of the XX-USB stack, the content of which is to be read. Valid types are 2, for the regular stack and 3, for the periodic (scaler) readout stack.

lpStackData
[out] Pointer to a variable array that receives the data returned by the operation, i.e., the content of a XX-USB stack.

Return Values
On success, the function returns the number of bytes read from XX-USB. The valid value is twice the length of the stack, as the latter stores 2-byte words.
Function returns a negative number on a failure.

6.10 xxusb_stack_execute

The xxusb_stack_execute function first, sends a buffer to XX-USB, causing the latter to interprete its content as a series of simple and complex CAMAC commands and to actually execute these commands and to write the returned CAMAC data to the USB port FIFO. Then, xxusb_stack_execute reads a buffer from XX-USB, containing the desired CAMAC data.

WORD xxusb_stack_execute{
HANDLE hDevice,
LPDWORD lpData,
};

Parameters
hDevice
[in] Handle to the XX-USB device.

lpData
[in] Pointer to a dual-use variable array. When calling the function, the array contains the data encoding the sequence of desired commands (CAMAC commands for CC-USB and VME commands for VM-USB) to be performed by XX-USB. The first element of the array is the number of bytes. The following command has to be defined similar to the CAMAC / VME command stack (see paragraph 4.5). Upon return, the array contains the CAMAC (CC-USB) or VME (VM-USB) data, respectively.

Return Values

On success, the function returns the number of bytes read from XX-USB. The valid value is twice the number of 16-bit data words returned plus 2 (CC-USB) or 4(VM-USB). The latter "overhead" bytes contain event terminator word (0xFF for CC-USB, and 0xFFFF for VM-USB).
Function returns a negative number on a failure.

6.11 xxusb_usbfifo_read

The xxusb_usbfifo_read function reads the content of the USB port FIFO of XX-USB or times out whenever this FIFO has not set the "FIFO Full" flag. **This function is using a slow data sorting algorithm and is kept for compatibility to earlier software. Please use the xxusb_bulk_read instead!!!**

```
WORD xxusb_usbfifo_read{
        HANDLE hDevice,
        LPDWORD lpData,
        WORD wDataLen,
        WORD wTimeout
};
```

Parameters

hDevice

[in] Handle to the XX-USB device.

lpData

[out] Pointer to a variable array that receives the data returned by the operation, i.e., the content of the USB port output FIFO of the XX-USB.

wDataLen

[in] Number of bytes to read. This number must be not less than the number of bytes stored in the output FIFO.

wTimeout

[in] Time in milliseconds, after which the I/O operation is canceled, should there be no data available for the readout.

Return Values

On success, the function returns the number of bytes read from XX-USB.
Function returns a negative number on a failure, which in most cases signifies a timeout condition.

Remarks

The xxusb_usbfifo_read is intended for use while XX-USB is in data acquisition mode. Upon timeouts, the host application receives the control and may reissue the command or terminate the acquisition

6.12 xxusb_bulk_read

The xxusb_bulk_read function reads the content of the USB port FIFO of XX-USB or times out whenever this FIFO has not set the “FIFO Full” flag.

```
WORD xxusb_bulk_read{
    HANDLE hDevice,
    CHAR *pData,
    WORD wDataLen,
    WORD wTimeout
};
```

Parameters

hDevice

[in] Handle to the XX-USB device.

pData

[out] Pointer to a character array that receives the data returned by the operation, i.e., the content of the USB port output FIFO of the XX-USB.

wDataLen

[in] Number of bytes to read. This number must be not less than the number of bytes stored in the output FIFO.

wTimeout

[in] Time in milliseconds, after which the I/O operation is canceled, should there be no data available for the readout.

Return Values

On success, the function returns the number of bytes read from XX-USB.
Function returns a negative number on a failure, which in most cases signifies a timeout condition.

Remarks

The xxusb_bulk_read procedure should be used for all stack mode DAQ read operations of CC-USB and VM-USB.

6.13 xxusb_bulk_write

The xxusb_bulk_write function writes a character array to the USB port FIFO of XX-USB.

```
WORD xxusb_bulk_write{
    HANDLE hDevice,
    CHAR *pData,
```

WORD wDataLen,
WORD wTimeout
};

Parameters
hDevice
[in] Handle to the XX-USB device.

pData
[out] Pointer to a character array that receives the data returned by the operation, i.e., the content of the USB port output FIFO of the XX-USB.

wDataLen
[in] Number of bytes to read. This number must be not less than the number of bytes stored in the output FIFO.

wTimeout
[in] Time in milliseconds, after which the I/O operation is canceled, should there be no data available for the readout.

Return Values
On success, the function returns the number of bytes read from XX-USB.
Function returns a negative number on a failure, which in most cases signifies a timeout condition.

Remarks
The xxusb_usbfifo_read is given for the sake of completeness.

6.14 xxusb_flashblock_program

The xxusb_flashblock_program function programs one sector of 256 bytes of the flash memory (FPGA configuration memory)

WORD xxusb_usbfifo_read{
HANDLE hDevice,
UCHAR *pData,
};

Parameters
hDevice
[in] Handle to the XX-USB device.
pData
[out] Pointer to the configuration (byte) data array.

Return Values

On success, the function returns the number of bytes written to XX-USB – the correct number is 518.
Function returns a negative number on a failure, which in most cases signifies a timeout condition.

Remarks
To program the flash memory, one must call repeatedly xxusb_flashblock_program, while pausing for at least 30ms between consecutive calls and incrementing the pointer to the data array by 256 on each consecutive call. The device must be in programming mode with the rotary selector in one of the 4 "P" positions.
The configuration file of a XC3S200 FPGA of CC-USB will occupy 512 sectors of flash memory (512 calls to the xxusb_flashblock_program). The XC3S400 FPGA of VM-USB will occupy 830 sectors of flash memory.

7 CC_USB SPECIFIC FUNCTIONS

The following functions are specific to the CC_USB. They are built on top of the general purpose functions described in section 6 and provide users with an eaiser and more transparent way of communicating with the controller.

7.1 CAMAC_register_write

The CAMAC_register_write function writes to the internal registers of the CC_USB as described in section 3.2.

```
short CAMAC_register_write{
        HANDLE hDevice,
        USHORT Address,
        LONG Data
};
```

Parameters
hDevice
[in] Handle to the XX-USB device.
Address
[in] Internal register address to write to, as specified in Section 3.2
Data
[in] Data to be written to the specified register

Return Values
On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

7.2 CAMAC_register_read

The CAMAC_register_read function read from the internal registers of the CC_USB as described in section 3.2.

```
short CAMAC_register_read{
        HANDLE hDevice,
        USHORT Address,
        LONG Data
};
```

Parameters

hDevice

[in] Handle to the XX-USB device.

Address

[in] Internal register address to read from, as specified in Section 3.2

Data

[out] Data read from the specified register

Return Values

On success, the function returns the number of bytes read from the CC-USB
Function returns a negative number on a failure

7.3 CAMAC_DGG

The CAMAC_DGG function allows the user to setup the characteristics of the Delay and Generator channels of the CC_USB.

```
short CAMAC_DGG{
        HANDLE hDevice,
        USHORT channel,
        USHORT trigger,
        USHORT output,
        USHORT delay,
        USHORT gate,
        USHORT invert,
        USHORT latch
};
```

Parameters

hDevice

[in] Handle to the XX-USB device.

channel

[in] The DGG channel you wish to modify. Valid values are:

0 – For DGG channel A

1 – For DGG channel B

trigger
[in] Determines the start of the DGG. Valid Values are:
0 – Channel Disabled
1 – NIM input 1
2 – NIM input 2
3 – NIM input 2
4 – Event Trigger
5 – End of Event
6 – USB Trigger
7 – Pulser

output
[in] Determines the NIM output used for the DGG channel. Valid values are:
1 – NIM O1
2 – NIM O2
3 – NIM O3

delay
[in] Sets the delay between the trigger and beginning of the gate in units of 10ns

gate
[in] Sets the length of the gate in units of 10ns

invert
[in] Determines whether or not the DGG is inverted. Valid values are:
0 – Not inverted
1 – Is inverted

latch
[in] Determines whether or not the DGG is latched. Valid values are:
0 – Not latched
1 – Is latched

Return Values
On success, the function returns 1
Function returns a negative number on a failure

7.4 CAMAC_LED_settings

The CAMAC_LED_settings function allows the user to setup the LEDs on the front panel of the CC_USB. Details about the LED settings are found in section 3.2.5

```
short CAMAC_LED_settings{
    HANDLE hDevice,
    USHORT LED,
    USHORT code,
    USHORT invert,
    USHORT latch
};
```

Parameters

hDevice
[in] Handle to the XX-USB device.
LED
[in] The LED you wish to modify. Valid values are:
1– RED
2 – GREEN
3 – Yellow
code
[in] Determines what event the LED is linked to. Valid valuesare 0-7 and are described in section 2.3.5
invert
[in] Determines whether or not the LED is inverted. Valid values are:
0 – Not inverted
1 – Is inverted
latch
[in] Determines whether or not the LED is latched. Valid values are:
0 – Not latched
1 – Is latched

Return Values
On success, the function returns the number of bytes from from the CC_USB.
Function returns a negative number on a failure

7.5 CAMAC_Output_settings

The CAMAC_Output_settings function allows the user to setup the NIM outputs on the front panel of the CC_USB. Details about the output settings are found in section 3.2.5

```
short CAMAC_Output_settings{
    HANDLE hDevice,
    USHORT Channel,
    USHORT code,
    USHORT invert,
    USHORT latch
};
```

Parameters
hDevice
[in] Handle to the XX-USB device.
Channel
[in] The NIIM output channel you wish to modify. Valid values are:
1 – O1
2 – O2
3 – O3
code

[in] Determines what event the output is linked to. Valid valuesare 0-7 and are described in section 2.3.5

invert

[in] Determines whether or not the output is inverted. Valid values are:

0 – Not inverted

1 – Is inverted

latch

[in] Determines whether or not the output is latched. Valid values are:

0 – Not latched

1 – Is latched

Return Values

On success, the function returns the number of bytes from from the CC_USB.
Function returns a negative number on a failure.

7.6 CAMAC_scaler _settings

Configures the internal CC-USB scaler (SelSource register)

```
short  CAMAC_scaler_settings{
       usb_dev_handle *hdev,
       short channel,
       short trigger,
       int enable,
       int reset}
```

Parameters:

hDevice

[in] Handle to the XX-USB device

channel

[in] The number which corresponds to the scaler channel:
0 - for Scaler A
1 - for Scaler B

code

[in] The Output selector code, valid values are listed in the manual

enable

[in] =1 enables scaler

latch

[in] =1 resets the scaler at time of call

Return Values

On success, the function returns the number of bytes from from the CC_USB.
Function returns a negative number on a failure.

7.7 CAMAC_write_LAM_mask

The CAMAC_write_LAM_mask function writes a LAM mask to the appropriate internal register of the CC_USB. This LAM mask is only used for DAQ-mode LAM based stack triggering

short CAMAC_write_LAM_MASK{
HANDLE hDevice,
LONG Data
};

Parameters
hDevice
[in] Handle to the XX-USB device.
Data
[in] LAM mask passed to CC_USB

Return Values
On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

7.8 CAMAC_read_LAM_mask

The CAMAC_read_LAM_mask function reads the LAM mask from the appropriate internal register of the CC_USB

short CAMAC_read_LAM_MASK{
HANDLE hDevice,
LONG Data
};

Parameters
hDevice
[in] Handle to the XX-USB device.
Data
[out] LAM mask read from the CC_USB

Return Values
On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

7.9 CAMAC_write

The CAMAC_write function writes a 24 bit word to a CAMAC address.

short CAMAC_write{
HANDLE hDevice,

```
        USHORT N,
        USHORT A,
        USHORT F,
        LONG Data,
        USHORT Q,
        USHORT X,
};
```

Parameters

hDevice

[in] Handle to the XX-USB device.

N

[in] CAMAC station number to write to

A

[in] CAMAC Sub-address to write to

F

[in] CAMAC Function for **F (16 ….23)**

Data

[in] data written to the CC_USB

Q

[out] Q response from CAMAC dataway

X

[out] comment accepted response from CAMAC dataway

Return Values

On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

7.10 CAMAC_read

The CAMAC_read function reads a 24 bit word from a CAMAC address.

```
short CAMAC_read{
        HANDLE hDevice,
        USHORT N,
        USHORT A,
        USHORT F,
        LONG Data,
        USHORT Q,
        USHORT X,
};
```

Parameters

hDevice

[in] Handle to the XX-USB device.

N

[in] CAMAC station number to read from

A
[in] CAMAC Sub-address to read from
F
[in] CAMAC Function for **F (0 … 15 and 24 … 31)**
Data
[out] data read from the CC_USB, **0 for F (8 … 15 and 24 … 31)**
Q
[out] Q response from CAMAC dataway
X
[out] comment accepted response from CAMAC dataway

Return Values
On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

7.11 CAMAC_Z

The CAMAC_Z function performs a CAMAC initialize.

```
short CAMAC_Z{
    HANDLE hDevice,
};
```

Parameters
hDevice
[in] Handle to the XX-USB device.

Return Values
On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

7.12 CAMAC_C

The CAMAC_C function performs a CAMAC clear.

```
short CAMAC_C{
    HANDLE hDevice,
};
```

Parameters
hDevice
[in] Handle to the XX-USB device.

Return Values
On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

7.13 CAMAC_I

The CAMAC_I function performs a CAMAC inhibit.

```
short CAMAC_I{
        HANDLE hDevice,
        USHORT inhibit
};
```

Parameters

hDevice

[in] Handle to the XX-USB device.

inhibit

[in] Determines whether CAMAC inhibit is on or off, Valid values are:

0 – Inihibit OFF

1 – Inhibit ON

Return Values

On success, the function returns the number of bytes written to CC-USB
Function returns a negative number on a failure

8 APPENDIX A: USE OF MULTIPLEXED USER DEVICES, FIRMWARE <5.01

The following section describes the features of the CC-USB for Firmware version below 5.01.

User LED and NIM Output Selectors – Read/Write

CAMACCAMAC A = 4 / 0x4 - LED Register
CAMACCAMAC A = 5 / 0x5 - NIM output register

Numbers stored in these registers identify sources of User LEDs and NIM Outputs. The actual selection of sources is firmware specific and subject to customization. The general bit composition of the selector word is shown in the table below

Yellow LED			Green LED			Red LED		
21	20	16-18	13	12	8-10	5	4	0-2
Latch	Invert	Code	Latch	Invert	Code	Latch	Invert	Code

NIM O3			NIM O2			NIM O1		
21	20	16-18	13	12	8-10	5	4	0-2
Latch	Invert	Code	Latch	Invert	Code	Latch	Invert	Code

The 3-bit code identifies the source of the signal. For firmware 95000101, the sources are as follows:

Code	Red LED	Green LED	Yellow LED
0	Event Trigger	Acquire	NIM I3
1	Busy	CAMAC F1	Busy
2	USB Trigger	Reserved	NIM I2
3	USB Out FIFO not empty	Event Trigger	CAMAC S1
4	USB In FIFO not full	CAMAC N	CAMAC S2
5	Reserved	Reserved	USB In FIFO not empty
6	Acquire	NIM I1	Executing scaler stack
7	CAMAC F2	USB In FIFO not empty	USB Trigger

Code	NIM O1	NIM O2	NIM O3
0	Busy	Event Trigger	End Of Busy
1	Event Trigger	CAMAC F1	Busy
2	USB Trigger	CAMAC N	NIM I2
3	DGG_A	Acquire	CAMAC S1
4	DGG_B	DGG_A	CAMAC S2
5	USB In FIFO not empty	DGG_B	DGG_A
6	Acquire	NIM I1	DGG_B
7	CAMAC F2	USB In FIFO not empty	USB Trigger

Note 1. “Busy” signal indicates that stack processing is in progress, with CAMAC operations not having completed. “Busy” is asserted when event readout is triggered and de-asserted as soon as CAMAC operations are completed.

Note 2. “Acquire” indicates that the data acquisition mode is active.

Note 3. “USB Trigger” is generated in response to writing to bit 1 of Action Register.

Note 4. “Event Trigger” indicates that event readout has been triggered.

Note 5. Invert bit causes the signal to be inverted

Note 6. Latch bit causes the signal to be latched. To release the latch one must toggle the bit.

Note 7. DGG_A and DGG_B are output pulses of the two user delay and gate generators.

CAMAC A = 7 / 0x7 - Delay and Gate Generator A
CAMAC A= 8 / 0x8 - Delay and Gate Generator B
CAMAC A= 13 / 0xD - Extended Delays (coarse)

The two Delay and Gate Generator Registers DGG_A and DGG_B as well as the DGG/P_A/B Extend register store data defining the length of the delay and the length of the gate in units of 10 ns (100 MHz clock) for either the gate and delay generator or for the pulser. These values can be set for channel A and B independently. The pulser is re-triggering after the defined delay time, i.e. the delay time + gate length defines the pulser repetition rate. With firmware revision 4.02 DGG_B has a 16bit and DGG_A a 24bit range for the delay setting The value of the delay for DDB_A is a composite of a high resolution value (10ns) and a coarse range value. Earlier firmware versions use only the fine (10ns) value.

Gate length = 10ns * Gate
Delay_A = 10ns * Delay_fine
Delay_B = 10ns * Delay_fine + 655.36μs* Delay_coarse
Pulser repetition period = Gate + Delay

DGG_A (A=7)

Bits	DGG_A 16-31	DGG_A 0-15
Function	Gate	Delay_fine

DGG_ B (A=8)

Bits	DGG_B 16-31	DGG_ B 0-15
Function	Gate	Delay_fine

DGG_ Ext (A=13)

Bits	(16-31)	DGG_A Ext 0-15
Function	Not used	Delay coarse (8bit)

9 APPENDIX B: USE OF MULTIPLEXED USER DEVICES, FIRMWARE <4.02

The FPGA configuration of the CC-USB may set up optionally various user devices, that are beyond the scope of a CAMAC controller, but which are intended to facilitate and reduce the cost of a data acquisition setup. The firmware of the CC-USB for **1.01> firmware <4.02** sets up two delay and gate generators, DGG_A and DGG_B and two 32-bit scalers, SCLR_A and SCLR_B.

9.1 Characteristics and the Use of Delay and Gate Generators

The two user gate and delay generators allow one to generate delays and gates in the range of 10 ns – approx. 650 us, with the 10 ns granularity.

To make use of an DGG_A or DGG_B, one simply needs to select the desired trigger signal by properly setting the respective selector code bits in the User Devices Register and set write the desired delay and gate data (in units of 10 ns) into the respective DGG register, as described in Section 3.2.7.

9.2 Characteristics and the Use of Scalers

The two user scalers allow one to count various signals and read out the resulting numbers in CAMAC-like commands. The latter commands access the scaler data without generating CAMAC bus activity. Both scalers are asynchronous with respect to the CC-USB clock, each using a dedicated fast clock network driven by the selected clock signal.

The use of the scalers is straightforward and entails selecting their respective input sources and enabling their operation by setting the respective “enable” bits. Optionally, one may wish to disable them by resetting the respective “enable” bits or clearing them by writing “1” to the respective “reset” bits.

User Devices Source Selector [Read/Write]

CAMAC A = 6 / 0x6

In addition to the two NIM outputs, firmware <5.01 implements four user devices - two delay and gate generators and two 32-bit scalers. All of these devices may use various signals as input/trigger signals, with the selection identified by respective code bits stored in the User Devices Source Selector register. Additionally, this register accommodates bits that enable and clear the two scalers. The bit composition of the User Devices source selector register is shown in the table below.

Device	Reset	Enable	Latch Bit	Invert Bit	Code
SCLR_A	5	4	-	-	0-2
SCLR_B	13	12	-	-	8-10
DGG_A	-	-	-		16-18
DGG_B	-	-	-	-	24-26

The meaning of the input selector codes is shown in the table below

Code	SCLR_A	SCLR_B	DGG_A	DGG_B
0	Disabled	Disabled	Disabled	Disabled
1	NIM I1	NIM I1	NIM I1	NIM I1
2	NIM I2	NIM I2	NIM I2	NIM I2
3	Event	Event	NIM I3	NIM I3
4	-	-	Event Trigger	Event Trigger
5	-	-	End of Event	End of Event
6	-	-	USB Trigger	USB Trigger
7	-	-	Pulser	Pulser

10 APPENDIX C: FIRMWARE < 1.01

CC-USB Architecture and User Interface for firmware < 1.01 (950000101)

The CC-USB presents to the user five internal devices or addresses shown in Table 1:
Table 1. Internal devices of CC-USB and their addresses

Address	Device
1	Register Block (RB)
2	CAMAC Data Readout Stack (CDS)
3	CAMAC Scaler Readout Stack (CSS)
4	CAMAC NAF Generator (CNAF)
5	Common Output Buffer

10.1 Register Block

The Register Block of CC-USB is composed of a number of registers identified by sub-addresses as shown in Table 2:

Table 2. Register sub-addresses and their functionality

Sub-address	Register	Note
0	Firmware ID	Read-only
1	Global Mode	Read/Write
2	Delays	Read/Write
5	Scaler Readout Frequency	Read/Write
6	User LED Source Selector	Read/Write
7	User NIM Output Source Selector	Read/Write
8	LAM Mask	24-bits, Read/Write
10	Action	Read/Write
12	CAMAC LAM	24-bits, Read-Only
15	Serial Number	11 bits, Read-Only
16	24 bit counter (I-2)	24-bits, Read-Only
18	Event counter (I-1)	24-bits, Read-Only

10.1.1 Firmware ID Register

This Firmware ID register identifies the acting FPGA firmware in four hexadecimal digits MYFR, where M and Y represent the month and year of creation, and F and R represent the firmware and revision numbers, respectively.

10.1.2 Global Mode Register

The global mode register has the following 16-bit structure:

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	-			Arbitr	WdgFreq			HeaderOpt	-	EvtSepOpt	-		BuffOpt			

The BuffOpt bits (0-2) define the output buffer length. Bit 3 controls the mode of buffer filling, such that 0 closes buffers at event boundaries and 1 allows spreading events across the adjacent buffers:

BuffOpt Value	Buffer Length (words)
0	4096
1	2048
2	1024
3	512
4	256
5	128
6	64
7	Single Event

The EvtSepOpt set the number of event terminator word (hexadecimal FFFF), such that EvtSepOpt=0/1 cause one/two terminator word/s written at the end of each event.

The HeaderOpt bit controls the structure of the buffer header, such that HeaderOpt=0 writes out one header word identifying the buffer type (bit 15=1 – watchdog buffer, bit 14=0 – data buffer, bit 14=1 – scaler buffer) and the number of events in buffer. When HeaderOpt = 1, the second header word is written out listing the number of words in the buffer.

The WdgFreq bits define the frequency at which the watchdog is forcing writing of output buffer during data acquisition. The three bit number represents the time interval in seconds, counting from the end of an event, after which the watchdog triggers when no new event has been observed.

The Arbitr Bit, when set to 1 activates CAMAC bus arbitration.

10.1.3 Delays Register

The delays register stores the desired trigger delay (from the start signal applied to the NIM input to the actual start of the CAMAC readout) – least significant 8 bits and the LAM timeout period – most significant 8 bits. Both delays are in units of us.

10.1.4 Scaler Readout Frequency Register

The Scaler Readout Frequency Register stores the number defining the frequency at which scalers are to be read out (scaler stack is executed) during the data acquisition. The stored value is equal to the number of data events separating the scaler readout events. When the value is zero, scaler readout is suppressed.

10.1.5 User LED and NIM Output Selectors

Numbers stored in these registers identify sources of User LEDs and NIM Outputs. The actual selection of sources is firmware specific and subject to customization. The general bit composition of the selector word is shown in the table below

	Yellow LED / NIM O3					Green LED / NIM O2					Red LED / NIM O1				
Bit	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	Latch	Invert	Code			Latch	Invert	Code			Latch	Invert	Code		

The 3-bit code identifies the source of the signal. The sources differ for different LEDs and NIM outputs, but they are the same between the LED and NIM targets (i.e., Red LED has the same sources as the NIM output O1, Green LED the same as O2, and Yellow LED the same as O3) For firmware 4503, the sources are as follows:

Code	Red LED	Green LED	Yellow LED
0	Event Trigger	Acquire	NIM I3
1	Busy	CAMAC F1	Busy
2	USB Trigger	Reserved	NIM I2
3	USB Out FIFO not empty	Event Trigger	CAMAC S1
4	USB In FIFO not full	Data Buffer Full	CAMAC S2
5	Reserved	Reserved	USB In FIFO not empty
6	Acquire	NIM I1	Executing scaler stack
7	CAMAC F2	USB In FIFO not empty	USB Trigger

Code	NIM O1	NIM O2	NIM O3
0	Busy	Event Trigger	EndOfBusy
1	Event Trigger	CAMAC F1	Busy
2	USB Trigger	Reserved	NIM I2
3	USB Out FIFO not empty	Event Trigger	CAMAC S1
4	USB In FIFO not full	Data Buffer Full	CAMAC S2
5	Reserved	Reserved	USB In FIFO not empty
6	Acquire	NIM I1	Executing scaler stack
7	CAMAC F2	USB In FIFO not empty	USB Trigger

Note 1. “Busy” signal indicates that stack processing is in progress, with CAMAC operations not being completed. “Busy” is asserted when event readout is triggered and deasserted as soon as CAMAC operations are completed.

Note 2. “Acquire” idicates that the data acquisition mode is active.

Note 3. “USB Trigger” is generated in response to writing to bit 1 of Action Register.

Note 4. “Event Trigger” indicates that event readout has been triggered.

Note 5. Invert bit causes the signal to be inverted

Note 6. Latch bit causes the signal to be latched. To release the latch one must toggle the bit.

10.1.6 LAM Mask Register

The LAM Mask Register is a 24-bit register that stores the LAM Mask defining what combination of LAM's triggers event readout during the data acquisition. When zero, the readout is triggered by a signal applied to the NIM input.

10.1.7 Action Register

Bit 0 of the Action Register activates data acquisition in list mode, when event readout is triggered either by a start signal applied to the User NIM input I1 or a combination of LAMs coinciding with the LAM mask.

Writing "1" to Bit 1 of the Action Register generates an internal signal of 150ns duration, called USB Trigger. The bit is, in actuality, never set, i.e., requires no resetting. This signal can be routed to user NIM output O1 or O3 and/or displayed on user Red or Yellow LED.

Bit 2 of the Action Register clears a number of internal registers and is intended for primarily for use during firmware debugging.

Bit 3 clears the Input 2 Scaler (24-bit).

10.1.8 Serial Number Register

The Serial Number Register is a Read-Only register containing the serial number of the CC-USB. The serial number can be also obtained during the initialization of the USB port, e.g., by calling the libxxusb library function xxusb_devices_find.