Super CMDs

CMD__BUFFER_START

CMD[7:0]	CMDI[7:0]	Dummy
0xF1	0x00	0x0000 → **return buffer carries length 0xhLLLL**

CMD_BUFFER_END → not mirrored

CMD	CMDI[7:0]	Fill[7:0]	
0xF2	0x00	0x00	

REFERENCE_WORD

CMD[7:0]	CMDI[7:0]	Reference
0x01	0x01	0x0000

All read cmds go to CMD endpoint

READ_LOCAL

CMD	CMDI[7:0]	Address
0x01	0x02	Address[15:0]

READ_LOCAL_BLOCK

CMD	CMDI[7:0]	Address
0x01	0x03	Address[15:0]
Block_length , max 768 Words		

WRITE_LOCAL

CMD	CMDI[7:0]	Address
0x02	0x04	Address[15:0]
Data[31:0]		

WRITE_RESET (higest reset, resets everything)

CMD[7:0]	CMDI[7:0]	Fill
0x02	0x06	0x0000
Data[31:0]		

Flow Control

Write to UDP port 0x8002

WRITE_DELAY, without CMD frame. Expects only one 32 bit UDP word

CMD	CMDI[7:0]	Delay
0x02	0x07	Delay[15:0]

CMDs executed from stack

START_STACK → is mirrored as

CMD	endpoint	Dummy
0xF3	0x01; returns stack	0x0000; returns length

START_STACK_NODATA; Returns no data to data stack $$MBS

CMD	endpoint	Dummy
0xF3	0x02; Suppress data output	0x0000; returns length

Implementation for instruction “QUIET” required for TRIVA

Write D16/32-A24/32

CMD	AM	Argument
Write: 0x23	A16: 0x29 A24: 0x39 A32: 0x09	Data Length: 0x0001 = 16 bit 0x0002 = 32 bit
Address[31:0]	16, 24 or 32 b its used by addressed module	
Data[31:0]	16 or 32 bits used by addressed module	

FIFO Read

READ D16/32-A24/32

CMD	AM	Argument
Read: 0x12	A16: 0x29 A24: 0x39 A32: 0x09	Data Length: 0x0001 = 16 bit 0x0002 = 32 bit
Address[31:0] 16, 24 or 32 b its used by addressed module		

Read MBLT/2ESST/BLT32/64-A32

CMD	AM	Argument
0x12	BLT:0x0F,0x0B,0x3F,0x3B MBLT:0x0C,0x08,0x38,0x3C ESST:{sp[1:0], 0x20[5:0]}	Max cycles (64k max) 0xXXXX MBLT: number of 64bit words ESST: number of 64bit words (strikes)
Address[31:0]		

Read MBLT64/ESST64-A32, swapped words at read

CMD	AM	Argument
0x13	MBLT: 0x0C,0x08,0x38,0x3C ESST:{sp[1:0], 0x20[5:0]}	Max cycles (64k max) 0xXXXX MBLT: number of 64bit words ESST: number of 64bit words (strikes)
Address[31:0]		

MEMORY Read

READ D16/32-A24/32 Only in combination with previous akku read and Dread-BLT

CMD	AM	Argument
Read: 0x32	A16: 0x29 A24: 0x39 A32: 0x09	Data Length: 0x0001 = 16 bit 0x0002 = 32 bit
Address[31:0] 16, 24 or 32 b its used by addressed module		

Read MBLT/2ESST/BLT32/64-A32

CMD	AM	Argument
0x32	BLT:0x0F,0x0B,0x3F,0x3B MBLT:0x0C,0x08,0x38,0x3C ESST:{sp[1:0], 0x20[5:0]}	Max cycles (64k max) 0xXXXX MBLT: number of 64bit words ESST: number of 64bit words (strikes)
Address[31:0]		

Read MBLT64/ESST64-A32, swapped words at read

CMD	AM	Argument
0x33	MBLT: 0x0C,0x08,0x38,0x3C ESST:{sp[1:0], 0x20[5:0]}	Max cycles (64k max) 0xXXXX MBLT: number of 64bit words ESST: number of 64bit words (strikes)
Address[31:0]		

Write Word to data

CMD	Fill[23:0]
0xC2	0x000000
WORD[31:0]	

STACK_END → not mirrored to data

CMD	Fill[23:0]
0xF4	0x000000

New control CMDs

Write_SPECIAL_WORD to data

CMD	Type[23:0]
0xC1	0x0 = time_stamp, 0x1 = accu[31:0],

Address_Inc_Mode → Memory read / FIFO read → OK

CMD	Fill
0xC3	0x000000 0= FIFO read, 1=memory_read

Is reset at stack start to FIFO mode

WAIT_CLK → ok

CMD	Fill
0xC4	0x000000; TIME[23:0] [clks]

SIGNAL_AKKU signal masked and shifted Accu → ok

lower 16 bits of masked shifted Akku→ pattern for 16 internal IRQs

CMD	Fill[23:0]
0xC6	0x00000000

Mask_Shift_Accu; first mask is applied, then rotated left

Can be set and modified at any time

CMD	Shift bits +8 (8+1 = shift left, number multiplied by 2
0xC5	0x000000 after mask rotate left by up to 31 bits
And-Mask[31:0]	

Set_Accu

CMD	Fill
0xC8	0x000000
Set value[31:0]	

Active until MBLT, BLT, Dread, Signal

Read_to_Accu A32 D16/32

CMD	AM	Argument
0x14	A16: 0x2D A24: 0x3D A32: 0x0D	Data Length: 0x0001 = 16 bit 0x0002 = 32 bit
Address[31:0] 16, 24 or 32 b its used by addressed module		

Active until MBLT, BLT, Dread, Signal

BLT → loops for value of accu, max values = Argument
MBLT → loops for value of accu, max values = Argument

Cycle Dread when accu just set with accu argument

Comp_Loop_Accu **(value) 0x12345678 (mode)**

CMD		
0xC7		Compare mode: **0x0 =eq,** **0x1 = lt,** **0x2 = gt ;** **loop to previous "read_to_accu" if false**
Compare value[31:0]		

If Read to Accu or set_accu is before MBLT,BLT,DRead, CMDs, the number of iterations in the Accu after masking and shifting is executed.

Alias table to connect Signals to IRQs
0x7000 to 0x701C;

Signals: 8 Aliases can be specified. Ie. signal=14 translates to IRQ8: 0x7000 = 14, 0x7002 = 8
0x7000 Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x7002 IRQ [1..16] (0=off)
0x7004 Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x7006 IRQ [1..16] (0=off)
0x7008 Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x700A IRQ [1..16] (0=off)
0x700C Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x700E IRQ [1..16] (0=off)
0x7010 Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x7012 IRQ [1..16] (0=off)
0x7014 Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x7016 IRQ [1..16] (0=off)
0x7018 Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x701A IRQ [1..16] (0=off)
0x701C Detected Signal pattern [15:0]. (0x0000= off) translates to:
0x701E IRQ [1..16] (0=off)

0x7020 IRQ [1..16] (0=off) Assigns an IRQ to all signals which are not assigned before.

Order of execution
1) Mask <compare><ROL><mask> → set mask for following Akku cmd
2) set Accu <Value> / load akku <Amode> <Dmode> <address>
3) MBLT
BLT
Dread_loop
Scan_Wait <compare_value> (compare with masked, shifted ACCU)

MBS

jetzt habe ich mir noch einmal die auslesesequenz unseres hoch aufloesenden vme tdc vftx angeschaut. die kritische und wohl bis jetzt nicht vom mvlc unterstuetzte prozedur laeuft folgendermassen:

```
status = *pl_status;        (pl_status = vme base + 0x14, am=9, a32 d32 )

event_laenge = (status & 0x3ff0) >>4;

for (i=0; i<event_laenge; i++)
{
   *pl_data++ = *pl_fifo_data; (pl_fifo_data = vme base + 0x1c, am=9, a32 d32 )
}
```

DATA Output FORMAT

Stack_Header {Number of Following Words[14:0]}
READ_CMD
    Data[31:0]

BLT, MBLT, 2eSST
    Header{ 15’d0,BERR_Flag,1’b1,Number of Following Words[14:0]}

Write Word
    Word[31:0]

STACK header:

CMD[7:0]	ERR[3:0]	Stack_No[3:0]	CTR_ID[2:0]	Length [12:0]
0xF3 / 0xF9	Cnt,Syntax, Berr, timeout			0x0001

0xF9, Cnt = 1 → stack header of parts if events has parts
0xF3, Cnt = 0 → .stack header of last part of event or for non partial events
Carries the 3 error bits

Cnt : continue bit; Event is continued in next partial event
ERR {Cnt,Syntax, Berr, timeout}

Berr is only shown when Berr at Dread, or Dwrite.
“regular” Berr at block read is only shown in the Block read header
At read or write with Berr, an additional Berr Word 0xFFFFFFFF is placed.

CTR_ID: controller id in multi crate operation, 0..7

Length: max 8k for partial event

Block header
Stack execution is continued at time out errors.

CMD[7:0]	ERR[3:0]	Reserved[3:0]	Reserved[2:0]	Length[12:0]
0xF5	Cnt, 0, Berr, timeout	0000	000	0x0001

ERR {Cnt, 0, Berr, timeout}
Cnt = Block is continued in next partial event

Stack error messages on EP0, only when stack data goes to EP1

CMD[7:0]	ERR[3:0]	Stack_No[3:0]	Length
0xF7	0, Syntax, Berr, timeout		0x0001
{stack_number[15:0], stack_line _to_fail[15:0]}			

Rate of error messages limited to 2k/s
ERR {0, Syntax, Berr, timeout}

CMD_Buffers:

CMD[7:0]	ERR[3:0]	Stack_No[3:0]	CTR_ID[2:0]	Length [12:0]
0xF1 / 0xF2	Cnt,0,0,0,0			0x0001

0xF1, Cnt = 0 → .stack header of last part of data or for non partial data
Carries the 3 error bits
0xF2, Cnt = 1 → stack header of parts if data has parts

Cnt : continue bit; Data is continued in next partial event

maximum part size 255 words.

Communication Example

CMD send to MVLC

```
0xF100 buffer_len[15:0]        # start of buffer ;
Place Super cmds: {cmd[15:0], Argument[15:0]}
0x0101 0001   # place reference word
        0x0204 AAAA                 # Write local
                0xDDDD DDDD         # Data
        0x0105 AAAA                 # read local
0xF200 0000   #end of buffer
```

CMD return

--------------- Ethernet specific ----------------

```
{0x0110,reserved[3:0],frame number_cmd[11:0], reserved[3:0],data_words[11:0]}
    {0x0111,header_pointer[10:0]==0}
----------------------------------------------------
           0xF100 LLLL       # start of buffer , length of buffer;
           0x0101 WWWW  # W=reference word
               0x0204 AAAA                # Write local
                       0xDDDDDDDD         # Data
               0x0105 AAAA                # read local
                       0xDDDDDDDD         # read datF
```

Data return

--------------- Ethernet specific ----------------

```
{reserved[3:0],frame number_dat[11:0],reserved[3:0],data_words[11:0]}
    {reserved[15:0],reserved[3:0],header_pointer[11:0] }
----------------------------------------------------
           0xF3TS LLLL       # start of buffer : Stack number, length of buffer;
               0xDDDDDDDD
               0xF5010002                 # BLT block, terminated with Berr
                       0xDDDDDDDD         # BLT data
                       0xDDDDDDDD
           0xF3TS LLLL       # start of buffer : Stack number, length of buffer;
               0xDDDDDDDD
               0xDDDDDDDD
           0xF3TS LLLL       # start of buffer : Stack number, length of buffer;
               0xF5000003                 # BLT regular termination
               0xDDDDDDDD                 # 3 BLT data following
               0xDDDDDDDD
               0xDDDDDDDD
```

L = Number of following words
S = stack number
T = timeout at VME-bus (0x0 ok, 0x1 = timeout)

Wrapping UDP Data

Header0: {0,0,Chan[1:0], Packet_number[11:0], CTR_ID[2:0],number_of_datawords[12:0]}
Header1: {UDP_time_stamp[18:0],Header_pointer[12:0]}

UDP_time_stamp: increment in 1ms steps → runs 17.5 minutes
CTR_ID = controller ID
Chan[1:0]. 0 = command mirror;
1= response from stack fifo EP0, (direct execution, errors)
2= data from stack EP1 (event data)

number of data words: number of words to follow the two header words.
Header_pointer: position directly behind the header words is 0.

Communication with stack ececution CMD:

CMD send data

```
{0x0100, bref[15:0]}  # start of buffer , reference;
        {0x0101 000, stack[3:0]}        # write stack
                0xC200 0000  # Write word to Stack (referenmce wor dto identify data packet)
                0x000000123
                0x230D 0002  # Write D16A32
                0x0000 000F  # Write address = 0xF
                0x1234 5678  # Word to write
        0xF100 0000  # Stack end
        {0x0102,0x000,EPoint,Stack[3:0]}   # execute stack , Endpoint to send
0x01FF 0000
```

CMD Return data = identical to send data: (To Check if UDP package has arrived)

```
{0x0110,frame number[15:0]}
    {0x0111,pointer_last_word[15:0]}  = 3
        {0x0100, ref[15:0]}# start of buffer , reference ; // CMD-Marker
           {0x0101, 0x000,stack[3:0]} # write stack
                0xC200 0000  # Write word to Stack (referenmce wor dto identify data packet)
                0x000000123
                0x230D 0002  # Write D16A32
                0x0000 000F  # Write address = 0xF
                0x1234 5678  # Word to write
           0xF100 0000       # Stack end
        {0x0102, 0x000,EPoint,stack[3:0]}   # execute stack   → Produces a data package
        0x01FF 0000  #end of buffer
```

UDP associated data package

```
{0x0110,frame number[15:0]}
    {0x0111,pointer to last word[15:0]}  = 3
        {0x0112,pointer to first event_header.[15:0]} (none → 0)  = 1 //DATA Marker
                {ref[11:0],stack[3:0], data_length[15:0]   = 1
                0x00000123    #reference word written to stack
```

Register map

Registers from 1 to 0x5FFF are 32bit registers.
* They can be accessed by super CMDs
* They only allow **D32 read and write** access from stack.

Write   0x0010 Master mode = 1;
Read   0x0010 -> {14’h0000, synchonised, master}

0x0400 USB frame gap[15:0] multiples of 8ns;
Default 20000 = 250us;

Trigger IO

0x0200 Trigger-IO, Timer; Trigger area has D16 = 2 byte address increments

0x0200 select_address[9:0]; {level[1:0], xxx ,unit[4:0]} // only lower 32 units in present impl.
4 levels, 32 units per level

0x0204 Reset_register_mask[11:0]: {timer[3:0],counter[7:0]}
0x6090 resets according to mask

0x0300 … 0x031E 32x registers[15:0] for unit
0x0340 … 0x05
3E 32x connects[15:0] for unit 0...32,

//**0x0380**…..0x03EE Input connects: {con,level[1:0],unit[7:0],outputs[3:0]}
// Data: 14 bit connection coordinate;
//**0x03F0** connection level: lev_pattern[3:0];

Units:
Type = 1, Input with GG
Type = 2, output with GG
Type = 3 Timer
Type = 4
VME_IRQ_input 1...7
Stack_Busy 0...7
Soft_trigger 0...7
VME_mon: AS*,DS0*, Dtack*
Type = 5 receive_sync_triggers

Type = 8, Gate generator
GG:
A0= INF{inputs[3:0],outputs[3:0],Type[7:0]}
A1 = delay
A2 = width
A3 = holdoff
A4 = invert_input (bit[0])

Type = 16, Coincidence + Pattern unit

Type = 31 Stack_start
Type = 32 internal_IRQ_out
Type = 33 Send_sync_triggerss, 4 inputs
Type = 64 Dig_Oszi, 8 inputs, trigger output for stack start

0x1000Trigger staistics (1k words)
stamps, rates, counts

// Stack has D32 bit = 4 byte address increments
Stack execution triggers

0x1100	stack0		RW		{IMM,trigger_type[2:0],sub_trigger[4:0]}
0x1104	stack1		RW		
0x1108	stack2		RW		
0x110C	stack3		RW		
0x1110	stack4		RW		
0x1114	stack5		RW		
0x1118	stack6		RW		
0x111C	stack7		RW		

Trigger type[2:0]:
- 0 = no trigger
- 1= IRQ, sub_trigger = 0...6 for IRQ1...7 (ROAK)
- 2= IRQ without IACK (RORA)
- 3= external Trigger (via IO) from trigger network
- 4= Timer underrun trigger

IMM=1 immediately execute this stack

Stack address index page

0x1200	stack0		RW		Only lower 16 bits valid, point to stack start.
0x1204	stack1		RW		
0x1208	stack2		RW		
0x120C	stack3		RW		
0x1210	stack4		RW		
0x1214	stack5		RW		
0x1218	stack6		RW		
0x121C	stack7		RW		

Start / stop

0x1300	start	2	RW		Bit 0: start = 1; Bit 1= stacks active
0x1304	Controller_id	3	RW		Is transmitted in F3/F9 word and USB header0

Example: when setting start bit to 0: poll start register until bit1 gets 0. Then all trigger requests and last stack have been processed.

0x2000 2k *32-bit Words = 4k-bytes Memory available for Stacks

0x2000	stack_start		RW	0	Stack is 32 bit oriented, → lower 2 address bits are not significant.
0x2004					

0x4000 to 0x43FF MDIO, 10 bits

Adress[9] = 1 → internal MAC access; 0 → PHY access

IP configuration 0x4400

Addr		bits		Def.	
0x4400	IP_low	16	RW		Own IP low
0x4402	IP_high	16	RW		Own IP high
0x440C	ip-cmd_low	16	RW		Dest ip-cmd_low
0x440E	ip_cmd_high	16	RW		Dest ip_cmd_high
0x4410	ip-dat_low	16	RW		Dest ip-dat_low
0x4412	ip_dat_high	16	RW		Dest ip_dat_high
0x4414	cmd_mac0		R		
0x4416	cmd_mac1		R		
0x4418	cmd_mac2		R		
0x441A	cmd_dest_port		R		
0x441C	data_dest_port		R		
0x441E	data_mac0		R		
0x4420	data_mac1		R		
0x4422	data_mac2		R		
0x4430	**jumbo_frame_on**	1	RW	0	Switch on jumbo frames (8k)

Own IP can be set by UDP via register or by DHCP
CMD dest IP can be modified when when MVLC is accessed at port 8000
Data dest IP can be modified when when MVLC is accessed at port 8001

VME module registers, Emulates a standard VME module at address 0xFFFFxxxx

Registers starting from 0x6000 are 16 bit register.
They can be written and read from stack by addressing A=FFFFxxxx, Data width is automatic
Adressing by direct cmd is also possible. Only lower 16 bits of data word valid.

SUPPORTED AM codes

2D A16 supervisory access
29 A16 non privileged data access

3D A24 supervisory data access
39 A24 non privileged data access

0D A32 supervisory data access
09 A32 non privileged data access

21 2eSST

3B A24 non privileged block transfer (BLT)
3F A24 supervisory block transfer (BLT)

0F A32 supervisory block transfer (BLT)
0B A32 non privileged block transfer (BLT)

3C A24 supervisory 64-bit block transfer (MBLT)
38 A24 non privileged 64-bit block transfer (MBLT)

0C A32 supervisory 64-bit block transfer (MBLT)
08 A32 non privileged 64-bit block transfer (MBLT)

2F CR/CSR access

Wrapping events

Header: {stack, event_length} → max event length is 32k Words.
    Data

further structure can be added by stack instructions. For example:
    Trigger time stamp,
    event number

Trigger registers:

Sends a status word
0x00 trigger counter_accept0
0x01 trigger counter_accept1
0x04 trigger counter_free0
0x05 trigger counter_free1
0x08 trigger time stamp0
0x09 trigger time stamp1

IRQ / Signaling concept:

V0018

IRQ 1...7 are VME IRQs (internal 0...6),
IRQ 8 to 15 (int 7..14) are internal IRQs
int(0..8) can be also connected to time stamper event FIFO
int(9) fixed DSO IRQ

V0019

IRQ 1...7 are VME IRQs (internal 0...6),
IRQ 8 to 15 (int 7..14) are internal IRQs
int(0..14) can be also connected to time stamper event FIFO
int(9) fixed DSO IRQ

Stack commands READ_SIGNAL and SIGNAL_WORD can emit a 16 bit internal signal
When not all 4 IRQ index registers are deactivated (== 0):
The pulsed signal is evaluated in 4 sets of 2 registers:
0x7000 IRQ index to be triggered (1...16) ; 0 means deactivated
0x7002 Compare signal. If equal, specified IRQ 0x7000 is emitted
.
.
0x700C
0x700E

When all IRQ indices are == 0; the signal is interpreted as direct IRQ trigger: bit 0 =irq1, bit 15 = IRQ16; IRQ9 is reserved for the DSO, and has no effect when set.