Interfacing mesytec VME modules

(Application note 001.3)

Scope of this application note is, to show the concept of data readout, and help users to integrate the mesytec modules into a data acquisition system.

Table of Contents

VME Overview

VME crates provides power supply and a high speed parallel bus for data transfer from and to up to 20 standard VME modules.

Data and Address Lines

Provides a parallel bus with **32 address** and **32 data lines**. In nuclear physics application the bus is controlled by one readout controller, which is the bus master. All digitizer modules are bus slaves.

The data lines are bidirectional and are used for reading and writing the addressed registers in the modules or reading the data.

The 16 high bits of the address lines are usually used to give the VME modules a unique base address in the 32 bit address space, while the lower 16 bits address the registers inside a module.

Lines to control the data flow

- There are **6 address modifiers** lines (AM).

  They are set by the controller and code for the tree basic data transfer modes of VME:

  1. single word data read/write.
  2. block transfer: up to 256 words with width of 32 bits can be read in one cycle from a module
  3. Multiplexed block transfer: read 256 words with a width of 64 bits from a module. In this case the address lines are also used for data transfer.

- Then there are **7 interrupt lines** to signal states from a module to the controller. So they are set by the modules. They can signal interrupts from 1 to 7. This is useful to signal to the controller that a module has data, or that a buffer is almost full, so anything which requires an action of the bus master.

  Which interrupt number to send is configured in the module registers also an 8 bit word, the interrupt vector, has to be define by register. When the bus master acknowledges an interrupt, the module sends an 8 bit word (the interrupt vector) down the VME bus line. When the controler recognizes this vector as the specified one, it starts its interrupt execution.

  The action to take after an interrupt, is left to the controller.

Addressing modes

The modules can be addressed by 24 bit addressing (A24) or by 32 bit addressing (A32).

The **base address**, which are the leading 16 bits in A32 mode, or the leading 8 bits in A24 mode, can be set via rotary coders on the motherboards of the modules or via address registers.

A second base address can be set by register for CBLT and MCST access

We suggest to used A32 addressing only.

Multi Cast (MCST)

Multicast means that data words can be written to more than one module simultaneously. The data transfer is identical to the standard data transfer, but addresses a second base address which is set common for more than one module. A good application is a readout reset to all modules after the data is read out.

Chained Block Transfer (CBLT)

To use this special readout, a second common base address (the same as for MCST) has to be defined for a group of modules. The modules have to be initialized from left (low slot number) to right as: "first module", several "mid modules" and one "last module". The bus controller can now read out those modules at the CBLT-base address as if it was only one module: one after the other module give their data to the bus, the last one emits a bus error, and terminates the data transfer this way. That's a very efficient way of readout, because the controller only has to start only one readout cycle and not one cycle per module.

For the most efficient FPGA powered controller, the advantage is moderate, as they need almost no time to start a cycle. For purely PC- controlled VME bridges, the advantage is significant.

Readout to Berr

For nuclear physics setups a very efficient readout mode was established. Every controller used in nuclear physics should be able to handle it.

As the amount of data to be read is usually unknown to the bus controller, it may just read with block transfers at the module FIFO. When there are no more data, or a full event is read out, the module emits a bus error "Berr". This is not an error condition, but simply a signal showing the termination of the data transfer. The controller can then directly proceed to the next module readout.

Many useful features like transferring a fixed number of events, can only be used when reading to Berr.

Address Modifiers recognized by all mesytec modules

AM codes, sent by the VME controller: (supervisory and privileged codes have the same effect)

For **standard read / write data transfer**, the following codes can be used:
0x3D A24 supervisory data access
0x39 A24 non privileged User data access
0x0D A32 supervisory data access
0x09 A32 non privileged data access

For **block transfer** the following codes can be used
0x3B A24 non privileged block transfer (BLT)
0x3F A24 supervisory block transfer (BLT)
0x0F A32 supervisory block transfer (BLT)
0x0B A32 non privileged block transfer (BLT)

For **multiplexed block transfer** the following codes can be used
0x3C A24 supervisory 64 bit block transfer (MBLT)
0x38 A24 non privileged 64 bit block transfer (MBLT)
0x0C A32 supervisory 64 bit block transfer (MBLT)
0x08 A32 non privileged 64 bit block transfer (MBLT)

Minimum requirements for mesytec modules

- VME-64 crate with +5 V, and ±12 V supply
- VME Controller / bridge: Interface from VME bus to the PC via USB, Ethernet...
  required: read/write D16, read D32, Addressing A24;
  useful: A32, BLT32, MBLT64, break data transfer on “Berr” (bus error).

Power requirement of the modules

Module	+5 V	+12 V	-12 V	power
MADC-32	190 mA	160 mA	80 mA	4 W
MQDC-32	1 A	300 mA	250 mA	12 W
MTDC-32	2.2 A	350 mA	520 mA	22 W
MDPP-16	2 A	100 mA	210 mA	14 W
MDI-2 *	230 mA	160 mA	100 mA	4.5 W

***** *power supplied to the MTM-16 devices on 2 buses is not included***

Readout modes

Event by event readout (mode 0)

The module buffers are only used for the data of one event. After conversion all modules are read out.
The procedure is:

Wait a fixed time for the slowest module to convert, then read out all modules.

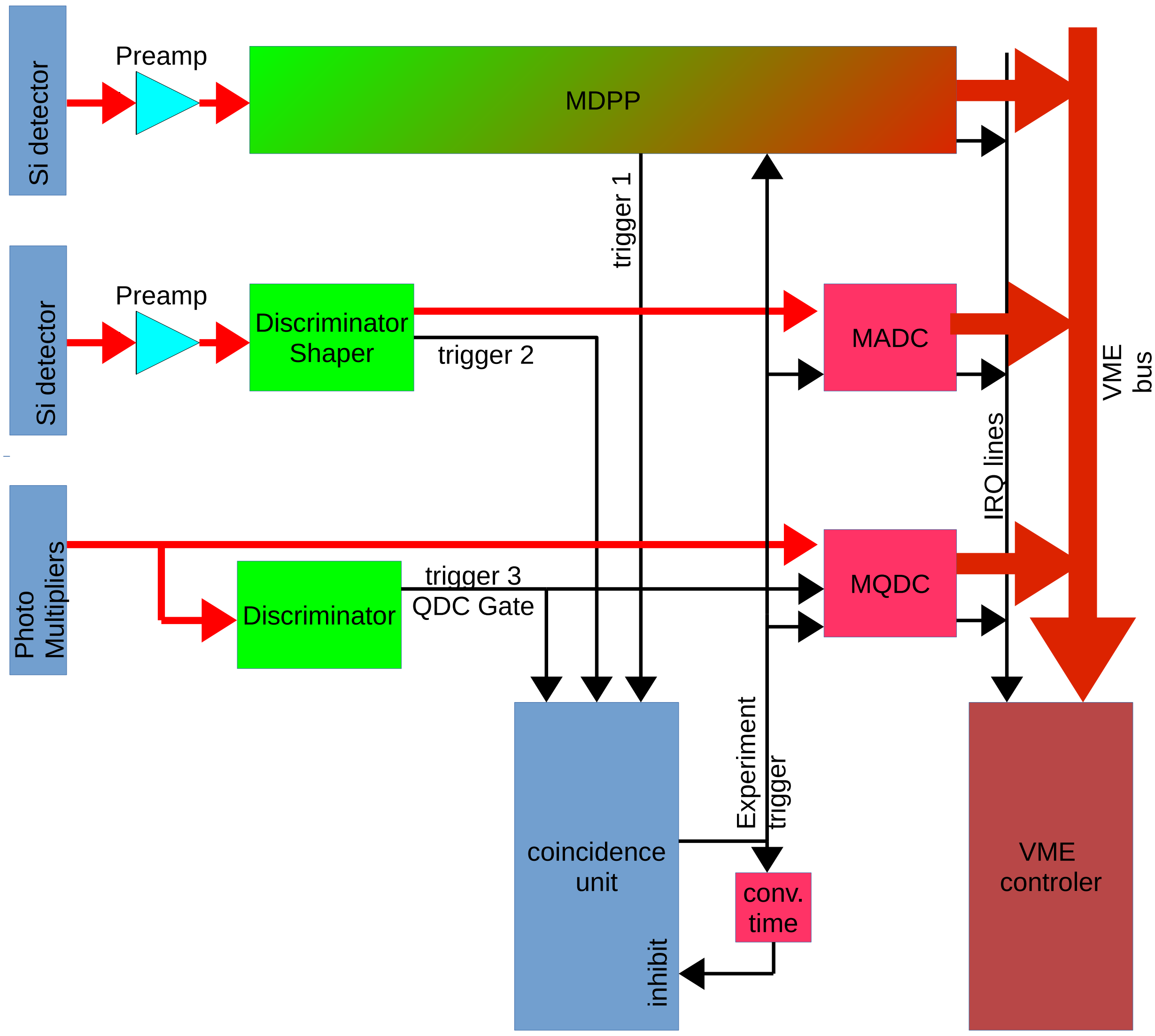
or

Wait for data ready lines of all modules get active, then start readout.

or

Wait for a data ready interrupt of the slowest module, then start readout.

Readout is terminated by writing event reset 0x6034 to all modules. After that, the modules are ready for the next trigger.
Options: reading data by CBLT is possible, event reset can be sent via multi cast.



Distributing the experiment trigger to all modules.

ECL Inputs can be chained to distribute a common experiment trigger to all types of mesytec modules.

If chaining the trigger this way, only the last module should have terminated inputs (via register setting).

The table shows the ECL inputs of all modules. The top ECL input is used for the common gate or trigger input. The second input can be used as gate /trigger when the 32 channel modules are operated in split bank, or in case of MTDC and MDPP, they are an additional timing trigger input.

If the event time stamp of the modules is not synchronized by the VME back plane clock (16 MHz), the sync input can supply an external clock to the modules. "Res" signal can then be used to reset the module clock counters simultaneously, but may also reset the event counter. The external reset is important when more than one VME-crate is needed or when modules of other manufacturers are synchronised in the setup. If all modules fit in one VME crate, it is easier to reset them simultaneously via multicast write to address 0x6090.

ECL direction	MADC		MQDC		MTDC		MDPP		
input	**T0**		**G0**		**T0**		**T0**		
input	**T1**	Sync	**G1**	Sync	**T1**	Sync	**T1**	Sync	Mon1
input	**Fast clear**	Res	**Exp. Trig**	Res		Res		Res	Mon0
	-		**-**		**-**		**T_Out**		
output	Busy		Busy		Busy		Busy		

In case of MQDC-32, G0 and G1 are the primary gates which control the signal integration gate. If MQDC is used with free running gates, converted events can be selected delayed by the event trigger input.

Multi event readout (mode 3)

In this case many events can run into the buffer. Data read out via VME is done all the time (asynchronous to the conversion).

For the fast mesytec modules, the total dead time, which is for event by event readout caused by conversion time, latency and VME data transfer time, is reduced to the pure conversion time. This may be an order of magnitude less than at event by event transfer.

If more than one module is read out, the events have to be synchronized.

Close synchronization

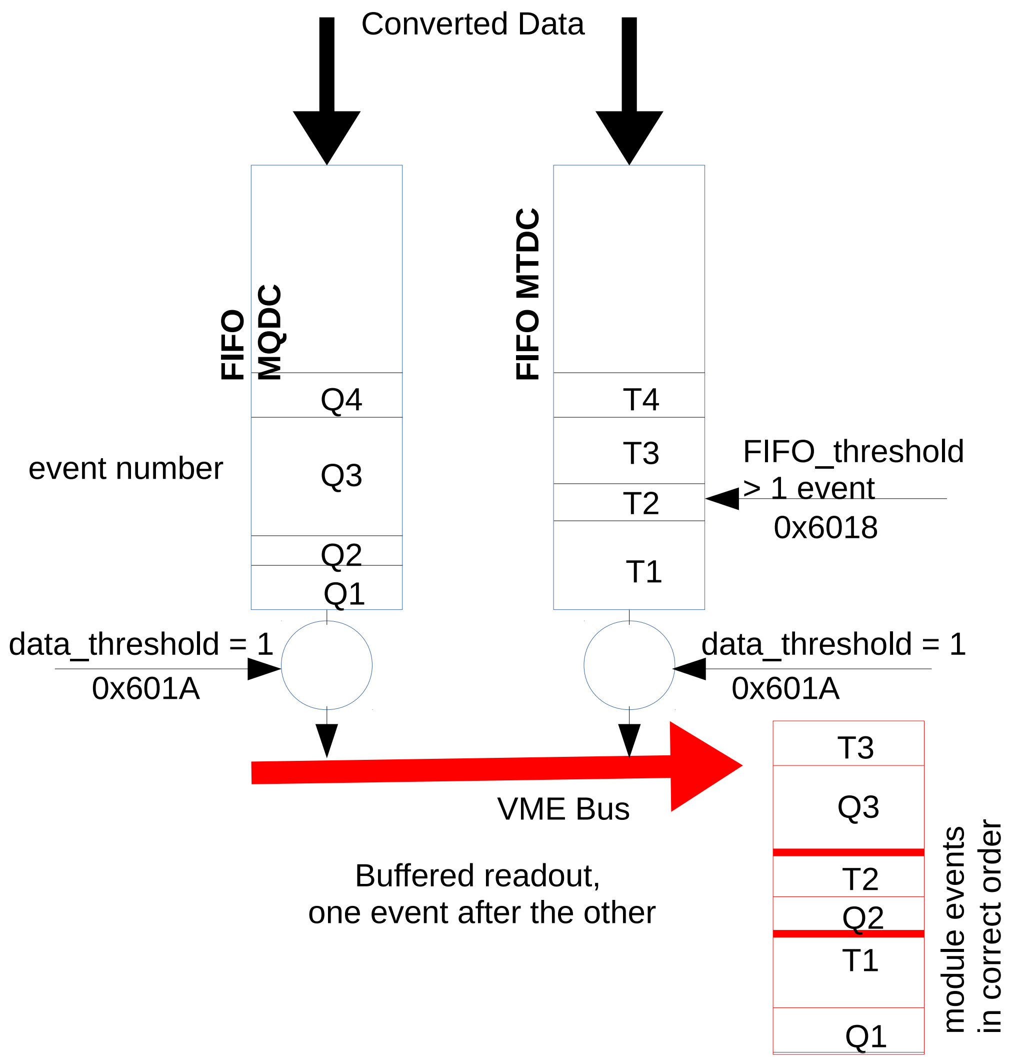
There are strict requirements for the trigger management. The same trigger has to be emitted to all modules. No module may ever miss a trigger, if this happens the whole data may get asynchronous and so corrupted. The trigger must be blocked for the conversion time of the slowest module (not for the latency + readout time !) else the trigger could get lost in one module.

For the fast mesytec modules the required trigger blocking time may be only 250 ns.

The buffers should not run empty at readout, so the first readout cycle should start when there are some events in the buffers. This can be done by signalling a certain fill level of one module to the DAQ. For example by signalling via busy output or via VME-IRQ. The FIFO threshold can be set via reg. 0x6018 "fifo_threshold". When set to 1 at least 1 event is in the buffer. When set to 40 at least 2 events are in the buffer.

Then one events is read out from all modules. Then waiting again for the fill buffer signal.

Transmission of one event is supported by the 0x601A "max_transfer_data" register. When set to 1, exactly one event is transferred before Berr is sent.



In the case of reading one event, the data structure is identical to the event by event readout, but the readout is asynchronous to the trigger, and the FIFO can be fully used.

It is not essential, but helps to detect data corruption, when time stamps or the event counter are added to the events.

When more than one VME-crate is required, time stamp or event counters are essential to synchronize the different data streams.

For VME controller with slow reaction time to interrupts or external readout triggers it may be useful to read out a larger fixed number of events from each module buffer. This mode is supported in the most actual firmware of all modules.

(MADC32 with FW0220, MQDC32 with FW0200, MTDC32 with FW0200)

Free running modules (mode 1 or 3)

This readout structure has no restrictions to the trigger management. The modules may get triggers from different sources at different rates.

The data buffers can be read out at any time in any order. Usually a high fill level in one module will initiate an irq or set an output line to prioritize the readout of that critical module. So data readout is fully asynchronous and concentrates on managing the fill level of the buffers.

A relatively complex software analyses the data stream and matches the events by time stamp.

So this readout mode is the fastest possible, has very low restrictions to the hardware, but needs sophisticated software and much more computer power.

Readout loop

1) start event readout at:

- VME-IRQ from module
- external trigger,
- poll register 0x6030 for data > 0

2) read at address 0x0000 via:

D32, BLT32, MBLT64

- until "BERR" = VME bus error; (fastest, most flexible solution)
- as many words as read in register 0x6030
- until register 0x6030 gets 0;
- until EOE is read from buffer

3) write 0x6034 (any value), go to 1)

Decode the data structure

Data are transmitted in 32 bit words. So the readout can be done by the VME read modes:

D32,
BLT32,
MBLT64 (two words transmitted at a time)

Filters to analyse the data structure of all mesytec VME modules:
Data are interpreted as a 32 bit unsigned integer

```
header_found_flag    = (data_integer & 0xC0000000)==0x40000000;
data_length          = (data_integer & 0x000003FF);              // maximum 10 bits
module_setting       = (data_integer & 0x0000FC00) / 0x400;      // 6 bits
                                                                 // depends on module
module_id            = (data_integer & 0x00FF0000) / 0x10000;    // 8 bits

data_found_flag      = ((data_integer & 0xF0000000)==0x10000000)   // MDPP format
                       || (data_integer & 0xFF800000)==0x04000000) // MADC,MQDC,MTDC…
channel_address      = (data_integer & 0x003F0000) / 0x0010000;  // 0 to 63
MDPP_flags           = (data_integer & 0x0FC00000) / 0x0040000;  // lower two bits
                                                                 // are under/overflow

extended_ts_flag     =  (data_integer & 0xFF800000)==0x04800000);
high_stamp           = (data_integer & 0x0000FFFF);              // 16 bits

fill word_found_flag =   data_integer==0x00000000;

eoe_found_flag             = (data_integer & 0xC0000000)==0xC0000000;
low_stamp            = (data_integer & 0x3FFFFFFF);              // 30 bits
```

Event counting

There are slight differences between the MADC-32 and MDI-2 and the other modules:

For MADC-32, MDI-2 the event counter is incremented, and then copied to the EOE word at the end of event buffer. So those modules start with event 1 after reset

The MQDC, MTDC, MDPP first copies the event counter to the EOE word, then increments it. So those modules start with event 0 after reset.

Module initialization, all mesytec modules

Basic initialization rules

the registers of mesytec modules are initialized at power up to values, which allow to run the module immediately in a basic mode.

Modifications to the basic power on mode can be added by modifying the corresponding registers.

If some sections are not needed, for example: multiplicity selection, IRQ handling, counter registers, Chained block transfer, Multicast., they should be left untouched.

Module initialization, standard registers

Address	Name	Bits	dir	default	Comment
	Address registers				
0x6000	address_source	1	RW	0	0 = from board coder, 1 from address_reg
0x6002	address_reg	16	RW	0	address to override decoder on board
0x6004	module_id	8	RW	0xFF	is part of data header If value = FF, the 8 high bits of base address are used (always board coder).
0x6008	soft_reset	1	W		breaks all activities, sets critical parameters to default. **Wait 200 ms after writing this register.**
0x600E	firmware_revision	16	R		Example: rev 1.4 = 0x0104

Identifying modules by software:
For all mesytec modules you can find ID information by reading the registers:

0x6008	# hardware ID
0x600E	# firmware revision

The result should be interpreted as hex code.
Hardware ID:

- MDI : 0x5001
- MADC-32: 0x5002
- MQDC-32: 0x5003
- MTDC-32: 0x5004
- MDPP-16: 0x5005
  - Firmware revision RCP: 0x1xxx
  - Firmware revision SCP: 0x2xxx
  - Firmware revision QDC: 0x3xxx
- VMMR-16: 0x5006
- MDPP-32: 0x5007
  - Firmware revision SCP: 0x2xxx

Blue: added features in firmware revision MTDC, MQDC FW0200 / MADC FW0220

	FIFO handling				
0x6036	multi event	4	RW	0	**mode = 0x0** single event readout (0x6034 clears event, allows new conversion) **mode = 0x3** multi event transfer. Transfer of data stopped ***after eoe word,*** when number of words specified in reg 0x601A is reached. So only full events are transferred. **mode = 0xb** multi event transfer. Transfer stopped after number of full events specified in 0x601A.
0x6038	marking_type	2	RW	0	0x0 → event counter 0x1 → time stamp 0x3 → extended time stamp

0x6018	fifo_threshold	16	RW	1	number of 32 bit words or number of events. (max words / events is module dependent) If data in the FIFO exceeds this threshold, - an IRQ can be emitted - a signal at busy out can be emitted
0x601A	Max_transfer_data	15	RW	1	Only for multi event mode 3. Number of words to transfer at buffer readout. Only full events are transmitted. 1 = minimum one data word → one full event is transmitted. 0 = transfer all data in buffer

0x601C	IRQ_source	1	RW	1	IRQ source: 0 = **event** threshold exceeded 1 = **data** threshold exceeded
0x601E	irq_event_threshold	15	RW	1	When the number of events in the FIFO exceeds this threshold, an IRQ is emitted.

0x603A	start_acq	1	RW	1	1 → start accepting triggers
0x603C	FIFO_reset		W		Initialize FIFO
0x6034	readout_reset		W		multi event = **0**: allow new trigger, allow IRQ At multievent = **3** or 0x**b:** clears Berr, allows next readout

** MADC-32: 8 k;   MQDC-32: 64 k,   MTDC-32: 48 k;   MDPP-16: 48 k   MDI-2: 1k

	Time stamping and reset				
0x6090	Reset_ctr_ab	2	RW		0x03 reset event- and time stamp counter
0x6096	time_stamp_source	2	RW	b00	bit0: sync source (VME=0, external=1) bit1: external reset enable = 1

Expert registers for all mesytec VME modules

Do not touch if IRQ is not needed

	IRQ (ROACK)				
0x6010	irq_level	3	RW	0	IRQ priority 1..7, 0 = IRQ off
0x6012	irq_vector	8	RW	0	IRQ return value
0x6014	irq_test	0	W		initiates an IRQ (for test)

Multicast and chained block transfer.
Do not touch if MCST CBLT is not needed

	MCST CBLT				
0x6020	cblt_mcst_control	8	RW	0	Reset CBLT status: 0x55 first module, MCST on: 0xA2 mid module, MCST on: 0x82 last module, MCST on: 0x8A
0x6022	cblt_address	8	RW	0xAA	A31..A24 CBLT- address
0x6024	mcst_address	8	RW	0xBB	A31..A24 MCST- address

Multiplicity filter

Completely suppresses events. Event counter counts up for suppressed events. Not suited for readout with close synchronization ! Wrong setting may deliver unpredicted data. Consider not to implement it.

MULT					
0x60B0	high_limit0	8	RW	16 /32/ 255	upper limit of responding channels (module dependent!)
0x60B2	low_limit0	8	RW	0	lower limit of responding channels

Control bus for all mesytec modules

Set I/Output for Cbus (lowest Lemo connector on the front panel)

In case the Cbus output and busy output are required, consider to configure the busy signal to the ECL output and convert it via external ECL to NIM converter.

Set the I/O for control bus operation:

MADC32, MQDC-32, MTDC-32

0x606E	NIM_busy	4	RW	0	0x3 → as Cbus output *(MADC-32: needs up to 100 us to get active)*

MDI-2

0x606E	Enable_busy	1	RW	1	0x0 → as Cbus output *(needs up to 100 us to get active)*

MDPP-16

0x6074	NIM0	4	RW	1	0x1= Cbus,

Cbus communication registers:

MRC	Module RC				
0x6080	rc_busno	2	RW	0	0 is external bus, comes out at busy output
0x6082	rc_modnum	4	RW	0	0...15 (module ID set with hex coder at external module)
0x6084	rc_opcode	7	RW		3 = RC_on, 4 = RC_off, 6 = read_id, 16 = write_data, 18 = read_data
0x6086	rc_adr	8	RW		module internal address, see box below
0x6088	rc_dat	16	RW		data (send or receive),write starts sending
0x608A	send return status	4	R		bit0 = active bit1 = address collision bit2 = no response from bus (no valid address)

Send time is 400 us. Wait that fixed time before reading response or sending new data.
Also polling at 0x608A for bit0 = 0 is possible

Read trigger, timing, event counter status

CTRA

Time stamp counters, event counters

All counters have to be read in the order: low word then high word !

They are latched at low word read. The event counter counts events which are written to the buffer.

CTRA 0x6090	**counters A**				
0x6090	Reset_ctr_ab	2	RW		b0001 resets all counters in CTRA, b0010 resets all counters in CTRB, b1100 allows single shot reset for CTRA with first edge of external reset signal. the bit bx1xx is reset with this first edge Reset of "counters A" will also reset the global 46 bit TDC time stamp
0x6092	evctr_lo	16	R	0	event counter low value
0x6094	evctr_hi	16	R	0	event counter high value
0x6096	ts_sources	2	RW	b00	bit0: frequency source (VME=0, external=1) bit1: external reset enable = 1
0x6098	ts_divisor	16	RW	1	time stamp = time / (ts_divisor) 0 means division by 65536
0x609C	ts_counter_lo	16	R		Time low value
0x609E	ts_counter_hi	16	R		Time high value

CTRB

Counters are latched when VME is reading the low word

Output value is divided by 40 to give a 1 us time basis

CTRB 0x60A0	**counters B**				
0x60A8	time_0	16	R		Time [1 us] (48 bit)
0x60AA	time_1	16	R		
0x60AC	time_2	16	R		
0x60AE	stop_ctr	2	RW	0	0 = run , 1= stop counter bit 0 all counter B bit 1 time stamp counter (A)

Built in pulsars to get first data

MADC-32

input range setting and gate generator setting have no effect

0x6070	pulser_status;	4	RW	0	0 = off, 4 = amplitude =0 5 = low amplitude (7 %), 6 = high amplitude (75 %) 7 = amplitude cycles: 0 -> low -> high → 0...

MQDC-32

0x6070	pulser_status;	4	RW	0	0 = off, 4 = amplitude = 0, 5 = use pulser-amplitude
0x6072	pulser_dac	8	RW	32	Pulser amplitude, typ. 32 for bin 2000

MTDC-32

0x6070	pulser_status;	1	RW	0	0 = off, 1 = on

MDPP-16

0x6070	pulsar_status;	1	RW	0	0 = Off, 1 = On
0x6072	pulsar_amplitude	12	RW	0	typical = 800

Module specific Initialization

MDPP-16

for software modules: SCP, RCP, CSI, QDC

Standard registers:

	Data output				
0x6042	tdc_resolution	3	RW	5	0 → 24 ps = 25 ns / 1024 1 → 49 ps = 25 ns / 512 2 → 98 ps = 25 ns / 256 3 → 195 ps = 25 ns / 128 4 → 391 ps = 25 ns / 64 5 → 781 ps = 25 ns / 32
0x6046	adc_resolution	3	RW	4	number of valid output bits for maximum amplitude 0 = 16 bits 1 = 15 bits 2 = 14 bits 3 = 13 bits 4 = 12 bits
0x605C	first_hit	1	RW	1	1 = only transmit first hit per channel in the window 0 = transmit all hits per channel in the window

	Trigger				
0x6050	win_start	15	RW	16k-16	win_start =(unsigned)((**time_delay_ns** / 1.56) + 0x3FFF) **time_delay_ns** in [ns], range: -25560 ns to +25560 ns; Start time widow of interest relative to trigger **time_delay_ns** negative: start before trigger, positive start delayed to trigger
0x6054	win_width	14	RW	32	win_width =(unsigned)(win_width_ns / 1.56) **win_width_ns** in [ns], range: 0 to +25560ns
0x6058	trig_source	10		0x100	**Defines the trigger which creates the window of interest**. • 0x001 : trigger 0 input • 0x002 : trigger 1 input • 0x100 : whole bank (self trigger, all channels) • or select a single channel: trig_source = 128 + (chan * 4); chan = channel number 0...15
0x605E	trigger_output	10	RW	0x100	**NIM1 trigger output.** • 0x100 : whole bank (OR of all channels) • or select a single channel: trig_output = 128 + (chan * 4); chan = channel number 0...15

	Inputs, outputs				**Sorted from top to bottom, as on the front panel**
0x6060	ECL3	8	RW	0x00	INPUT 0x00 = off 0x01 = trig0, terminated; 0x11 = trig0, unterm.
0x6062	ECL2	8	RW	0x00	INPUT 0x00 = off 0x01 = sync, terminated; 0x11 = sync, unterm. ; 0x02 = trig1, terminated; 0x12 = trig1, unterm. ; **when sync is selected also set reg 0x6096 !!**
0x6064	ECL1	8	RW	0x00	INPUT 0x00 = off 0x01 = reset terminated; 0x11 = reset, unterm. ;
0x6066	ECL0	4	RW	0x0	OUTPUT 0x0 = Off, 0x4 = Busy; 0x8 = data in buffer above threshold 0x6018 (= Data ready)
0x6068	NIM4	2	RW	1	INPUT 0x0 = Off, 0x1= Trig0_in
0x606A	NIM3	2	RW	0	Front panel Monitor setting overrides this setting. Then it is gets "mon 1" output. 0x0 = Off, 0x2 = Sync in **when sync is selected also set reg 0x6096 !!**
0x606C	NIM2	2	RW	1	Front panel Monitor setting overrides this setting. Then it is gets "mon 0" output. 0x00 = Off, and 0x01 = Trig1_in, 0x02 = Reset
	NIM1				Always trigger output
0x6074	NIM0	4	RW	1	0x0= Off, 0x1= Cbus, 0x4 = Busy_out (= FIFO full or ACQ stopped) 0x8 = data in buffer above threshold 0x6018 (= Data ready)

Front end setting, signal processing (RCP, SCP software module)

RCP: readout of reset charge integrating preamps,

SCP: readout of standard charge integrating preamps (RC feedback)

Channel addressing

0x6100	select_chan_pair	4	RW	8	channel to be modified: 0..7 channel pairs; chan 0,1 = 0, chan 2,3 = 1, ... 8 = all channels (set to common values)

Channel setting

	Parameter.			default	All times are in multiples of 12.5 ns
0x6110	**TF_int_diff:**	7	RW	2	common for 2 channels TF- integration/differentiation time, chan 0/1 valid values 2...125 (25 ns to 1.6 us)
0x6112	**PZ0:**	16	RW	0xffff	Not used for RCP module. signal decay_time0, channel 0 (for PZ compensation) valid: 64...64k (65535), 0.8 us to 800 us, and infinite
0x6114	**PZ1:**	16	RW	0xffff	Not used for RCP module. signal decay_time1, channel 1
0x611A	**Gain:**	14	RW	200	common for 2 channels, gain x 100 gain 1...250, chan 0/1; setting 100, gain=1 … setting 25000, gain = 250
0x611C	**threshold0:**	16	RW	0x00ff	0 to 64k (65535) . 64 k corresponds to full range.
0x611E	**threshold1**	16	RW	0x00ff	
0x6124	**Shaping_time**	11	RW	0x64	common for 2 channels , FWHM-width values 8...2000 (= 100 ns to 25 us)

Expert registers for MDPP-16 front end setting, signal processing

6126	**BLR**	2	RW	2	common for 2 channels, Base line restorer setting, 0 = off, 1 = soft (int. time = 8 shaping times), 2 = strict (int. time = 4 shaping times)
6128	**reset_time**		RW	16	common for 2 channels (min 16 = 200 ns)
612A	**signal_rise_ time**	7	RW	0	common for 2 channels • default = 0, for Si-detectors, constant rise time detectors -> shortest dead time • for **germanium detectors** with position dependent rise time, set to largest possible signal rise time. This results in highest resolution and ballistic loss correction. Determines flat top of triangular shaper pulse.

MADC-32

	operation mode				
0x6040	bank_operation	2	RW	0	b00 → banks connected b01 → operate banks independent b11 → toggle mode for zero dead time (use with internal gate generators enabled)
0x6042	adc_resolution	3	RW	2	0 → 2 k (800 ns conversion time) 1 → 4 k (1.6 us conversion time) 2 → 4 k hires (3.2 us conversion time) 3 → 8 k (6.4 us conversion time) 4 → 8 k hires (12.5 us conv. Time)

	gate generator				
0x6050	hold_delay0	8	RW	20	0= 25 ns, 1= 150 ns, then multiple of 50 ns
0x6052	hold_delay1	8	RW	20	same as for bank 0
0x6054	hold_width0	8	RW	50	multiple of 50 ns
0x6056	hold_width1	8	RW	50	Same as for bank 0
0x6058	use_gg	2	RW	0	01 = use GG0 10 = use GG1 (GG1 can only be activate when reg 0x6040 ! = 00, banks not connected)

IO 0x6060	**Inputs, outputs**				
0x6060	input_range	2	RW	0	input range: 0 → 4 V, 1 → 10 V, 2 → 8 V
0x6062	ECL_term	3	RW	b000	switch ECL terminators on (1= on) low bit for: "gate0", high bit for "fc" Switch terminators off when inputs are not used. Then inputs will be set to a well defined state by internal weak resistors.
0x6064	ECL_gate1_osc	1	RW	0	0 → gate1 input, 1 → oscillator input (**also set 0x6096 !!**)
0x6066	ECL_fc_res	1	RW	0	0 → fast clear input, 1 → reset time stamp oscillator input
0x6068	ECL_busy	1	RW	0	0 → as busy output, 1 → reserved
0x606A	NIM_gat1_osc	1	RW	0	0 → gate1 input, 1 → oscillator input (**also set 0x6096 !!**)
0x606C	NIM_fc_reset	1	RW	0	0 → fast clear input, 1 → reset time stamp oscillator, hold at value 0
0x606E	NIM_busy	4	RW	0	0 → as busy 1 → as gate0 output 2 → as gate1 output 3 → as Cbus output (needs up to 100 us.) 4 → buffer full 8 → data in buffer above threshold 0x6018

MQDC-32

	operation mode				
0x6040	bank_operation	3	RW	b000	0 → banks connected 1 → operate banks independent
0x6044	Offset_bank_0	8	RW	undef	Typical value: 130 Channels 0 to 15: Range 0 to 255, Shifts 4 k spectrum by ±1000 bins
0x6046	Offset_bank_1	8	RW	undef	same for channels 16 to 31

	gate limit				→ see lookup table at next page
0x6050	limit_bank_0	8	RW	255	4 = 4.5 ns, 255 = no limitation
0x6052	limit_bank_1	8	RW	255	same as for bank 0

	experiment trigger				
0x6054	exp_trig_delay0 (1)	14	RW	0	delay of experiment gate relative to end of QDC-gate (ns). Up to 16384 ns (16 k) For bank0 or both at joined bank
0x6056	exp_trig_delay1 (1)	14	RW	0	For bank 1 if not joined (1) only available in firmware revision FW0110 and higher, all devices upgradable

	Inputs, outputs				ECL and NIM IOs are counted from the bottom to the top from 0 to 3
0x6062	ECL_term	5	RW	b1100 0	switch ECL/LVDS terminators on (1= on) **Bit** | **4** | **3** | **2** | **1** | **0** ECL input term. | gates bank1 | gates bank0 | reset *ECL2* | gate1 ECL1 | gate0 ECL0
0x6064	ECL_gate1_osc (ECL2)	1	RW	0	0 → gate1 input, 1 → oscillator input (**also set 0x6096!!**)
0x6066	ECL_FC_Reset (ECL1)	2	RW	1	0 = Fast clear input (= "not Gate") 1 = Reset time stamp counter 2 = input for experiment trigger (1)
0x6068	Gate_select	1	RW	0	0 → Gate 0 and 1 from NIM-inputs, 1 → Gate 0 and 1 from ECL-inputs
0x606A	NIM_gat1_osc (NIM2)	2	RW	0	0 → gate1 input, 1 → oscillator input (**also set 0x6096!!**)
0x606C	NIM_FC_Reset (NIM1)	2	RW	0	0 = Fast clear input (= "not Gate") 1 = Reset time stamp counter 2 = input for experiment trigger (1) (1) only available in firmware revision FW0110 and higher, all devices upgradable

0x606E	NIM_busy (NIM0)	4	RW	0	0x0 → as busy (in independent bank operation: active when both banks are busy) 0x3 → as Cbus output 0x4 → buffer full 0x8 → data in buffer above threshold 0x6018

look up table for gate limit (reg 0x6050, 0x6052)

DAC	time [ns]
4	4,5
5	4,6
6	4,6
7	4,7
8	4,8
9	4,8
10	4,9
11	5,0
12	5,1
13	5,2
14	5,3
15	5,4
16	5,6
17	5,7
18	6,1
19	7,6
20	8,7
21	9,4
22	10,1
23	10,9
24	11,6
25	12,3

DAC	time [ns]
30	16
35	19
40	24
50	32
60	41
70	51
80	62
90	73
100	86
110	101
120	117
130	136
140	158
150	185
160	219
170	268
175	300

MTDC-32

	operation mode				
0x6044	output_format	1	RW	0	0 = standard (time difference) 1 = time stamper mode **when 1 is selected, further "operation mode" and "trigger" registers have no effect**
0x6040	bank_operation	1	RW	0	0 → banks connected 1 → operate banks independent
0x6042	tdc_resolution	5	RW	0	9 → 500 ps 8 → 250 ps 7 → 125 ps 6 → 62.5 ps 1 ns / 16 5 → 31.3 ps 1 ns / 32 4 → 15.6 ps 1 ns / 64 3 → 7.8 ps 1 ns / 128 2 → 3.9 ps 1 ns / 256
0x605C	first_hit	2	RW	b11	bit0 = bank0, bit1 = bank1 1 = only transmit first hit of a channel 0 = transmit all hits of a channel in the window

	Trigger				
0x6050	bank0_win_start	15	RW	16k-16	Unit: ns Start window of interest from trigger start, Offset +16 k = 16384; 16k → no delay win_start < 16 k, window starts before Trigger win_start > 16 k, window is delayed
0x6052	bank1_win_start	15	RW	16k-16	same as for bank 0
0x6054	bank0_win_width	14	RW	32	Unit: ns, max 16 k = 16 us, unsigned
0x6056	bank1_win_width	14	RW	32	Unit: ns
0x6058	bank0_trig_source	10		1	at split banks for bank0, else for both banks. **Defines the trigger which creates the window of interest**. This can be: one or both of the trigger inputs, any of the 32 channel inputs, a whole bank. • 0x001 : trigger 0 input • 0x002 : trigger 1 input • 0x100 : bank0 • 0x200 : bank1 • or select a single channel: trig_source = 128 + (chan * 4); chan = channel number 0...31
0x605A	bank1_trig_source	10		2	only at split bank, bank 1. Same parameters as above.

	Bank input configuration				
0x6060	Negative_edge	2	RW	b00	bank[1:0] flag, * for differential input jumper: 1 = channel trigger on negative edge * for unipolar input jumper: 1 = trigger on rising edge of the signal
0x6078	bank0_input_thr	8	RW	105	Discriminator level for unipolar input bank 0, For NIM 105 is optimum.
0x607A	bank1_input_thr	8	RW	105	Discriminator level bank1

IO 0x6060	**Inputs, outputs**				
0x6062	ECL_term	3	RW	b000	switch ECL/LVDS terminators on (1 = on) **Bit** | **2** | **1** | **0** ECL input term. | reset | gate1 | gate0
0x6064	ECL_trig1_osc	1	RW	0	0 → trig1 input, 1 → oscillator input (**also set 0x6096 !!**)
0x6068	Trig_select	1	RW	0	0 → Trigger 0 and 1 from NIM-inputs, 1 → Trigger 0 and 1 from ECL-inputs
0x606A	NIM_trig1_osc	2	RW	0	0 → trig1 input, 1 → oscillator input (**also set 0x6096 !!**)
0x606E	NIM_busy	4	RW	0	0x0 → as busy (= FIFO full or ACQ stopped) 0x3 → as Cbus output 0x8 → data in buffer above threshold 0x6018 (= Data ready)

MDI-2

	Trigger				
0x6040	seq_enable	2	RW	3	enable sequencer 1,0 or both
0x6042	trig_source0	3	RW	7	select trigger to start bus0 sequencer source: common, trigger1, trigger0
0x6044	trig_source1	3	RW	7	select trigger to start bus1 sequencer source: common, trigger1, trigger 0
0x6046	com_trig_source	2	RW	3	select source for common trigger I/O source: trigger1, trigger 0
0x6048	gen_trigger	1	W		generates software trigger, 0 = bus0, 1=bus1 (for test)
0x604A	gen_event	1	W		generate trigger and event data,0 = bus0, 1=bus1. Amplitude rising from 100 to 4 k (for test)
0x604C	veto_gate	1	RW	0	0 = use "veto" input as **veto** veto is possible from trigger to 25 ns before end of hold-delay. 1 = use "veto" input as **gate** (when active triggers are accepted) veto and gate have to be active at least 25 ns before hold_delay0/1 runs out. If different delays for bus1/0 are used, the shorter one is relevant. (Typically 475 ns after trigger for MTM-16 timing)

	hold / gate generation				
0x6050	hold_delay0	12	RW	1000	multiple of 0.5 ns (default: gate on bus0 starts 0.5 us after trigger)
0x6052	hold_delay1	12	RW	1000	same for bus 1
0x6054	hold_width0	7	RW	44	multiple of 25 ns (gate length on bus 0 default = 1.1 us)
0x6056	hold_width1	7	RW	44	same for bus 1

	Sequencer timing				
0x6060	bus_watchdog	1	RW	1	bus watchdog. 1 = on if no event trigger for 1 s, sends front end reset. Protects against deadlocks.
0x6062	frontend_reset	0	W		frontend_reset (sends "reset sequencer" to MTM-16, settings left unchanged)

0x6064	seq_clk_freq0	3	RW	3	clock frequency, Bus 0: 0 = 1.25 MHz, 1 = 2.5 MHz, 2 = 5 MHz, 3 = 10 MHz
0x6066	seq_clk_freq1	3	RW	3	clock frequency, Bus 1
0x6068	seq_busy	2	R		read sequencer 1 / 0 busy when seq ready (= 0) , data are available at buffer
0x606A	sample_delay_reg0	4	RW	3	one additional count delay for multiple of 2.5 m cables. For standard cable of 3 m set to 4
0x606C	sample_delay_reg1	4	RW	3	"
0x606E	enable_busy	1	RW	1	enable busy signal at Lemo output Do not allow when used as control bus or for time stamping

	Sequencer length				
0x6074	seq0_cct	10	RW	17	17 counts per MTM-16 needed
0x6076	seq1_cct	10	RW	17	17 counts per MTM-16 needed
0x6078	allow_sync_word	1	RW	0	allow_sync_words in FIFO buffer. Useful when zero suppression in front end MTM-16 (future feature)

Examples

Example 1, basic initialization and readout

Readout event by event, interrupt driven

Initialization:

```
  D16: 0x6010 = 1         // use IRQ 1
```

module specific initialization, here: MADC

```
  D16: 0x603C = any value  // FIFO reset
  D16: 0x6034 = any value  // event reset
```

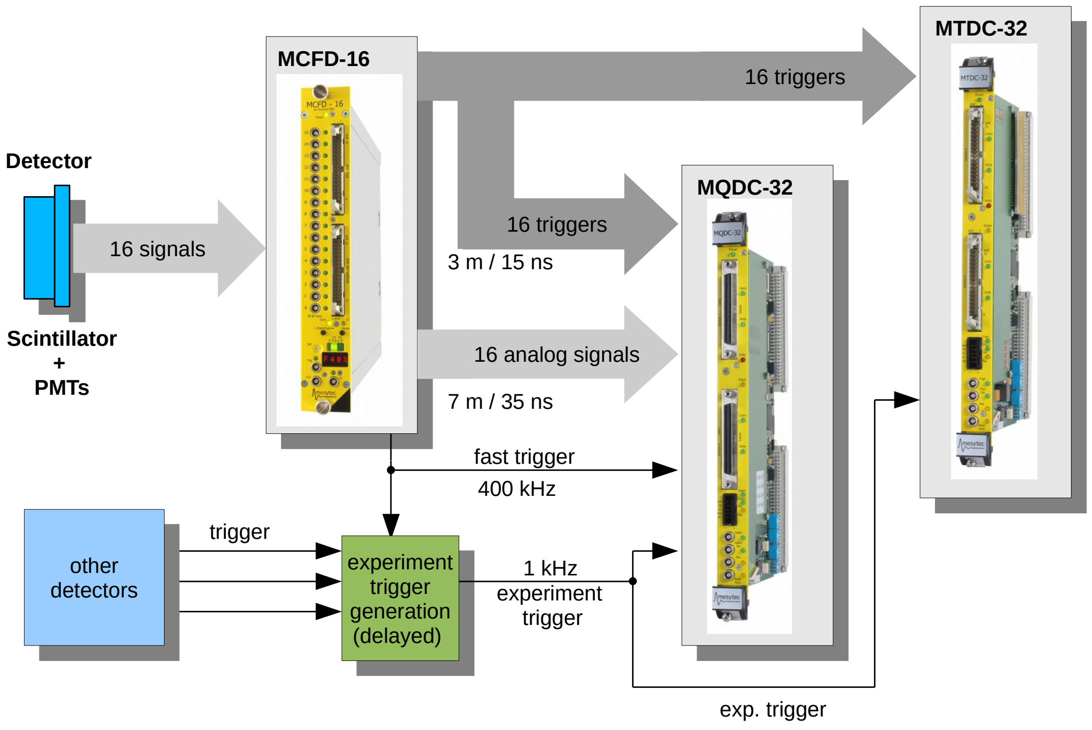
readout:

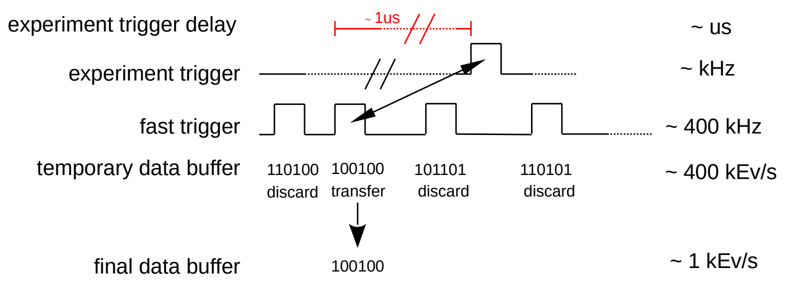
1) wait for interrupt
2) read BLT32, maximum block length (runs until BERR)
3) D16: 0x6034 = any value, then go to 1)

Module specific initialization → module chapters

Example 2, setup MCFD & MQDC

Signal processing of a 16 channel scintillator , light converted with a multi anode photo multiplier.





The 16 channel from PMT are connected to the inputs of an MCFD-16 discriminator.

The fast OR output is directly used as a free gate for MQDC-32 (direct short Lemo connection). This helps to keep the required cable delays as short as possible.

Trigger 1..3 output of the MCFD-16 can be used to create an experiment trigger, which is fed to the experiment trigger input of the MQDC-32 and is also used as trigger for the MTDC-32. The delay of experiment trigger is not critical.

This can also be done with many modules. In this case the fast OR from MCFDs feed the individual gate inputs (best is NIM inputs) of the MQDC-banks (operate MQDC in split bank).

The common experiment trigger can be distributed to MTDC-32 via ECL -T0 input and to the MQDC vial ECL experiment trigger input (3. from top).

Readout is done event by event.   In the present firmware the experiment gate does only produce an event data structure and increments the event counter when coincident converted data are found. This will be modified in the next firmware revision. Then an empty frame is sent when no conversion is found.

Connection: The QDC ECL outputs are wired to the QDC gate inputs and then to the TDC inputs. So QDC gate inputs should be unterminated, TDC inputs are terminated (TDC input jumpers set to "terminated").

The Analog signals from CFD output are wired to the analog input of the QDC. CFD output and QDC input should be differential, the QDC input jumpers must be differential and set to negative polarity. The cable length must be adapted to the required delay of the analog signal.

The experiment trigger is generated from signals of all CFDs and maybe triggers of other detectors, and is the converted to the main trigger in ECL. This signal is distributed to all TDCs and QDCs. As the experiment gate of QDCs do not require a very strict timing, it may also be a second driver output of the event gate.

The QDC event gate should have the length required for the coincidence in the experiment, but at least 25ns.

Register setup MQDC:

All IOs are counted from bottom to top, starting with 0.

Thresholds are not required, when individual gates from CFD are used. Then the CFD thresholds determine when a gate is created and QDC conversion is started.

IO setting

Set NIM3 = gate0, NIM2 = gate1

0x6068 = 0 //Gate_select → Gate 0 and 1 from NIM-inputs, Lemo3 is gate0 by default

0x606A = 0 // NIM_gat1_osc, Lemo2 is → gate1 input

Set ECL1 for Experiment gate.

0x6066 = 2 // ECL_FC_Reset -> ECL1 is input for experiment gate

ECL terminators: For all MQDC modules except the last one

0x6062 = b00000 = 0 // ECL_term: ECL1..3 all inputs unterminated

For the last module, terminate Experiment gate input:

0x6062 = b00100 = 4 // ECL_term: ECL2 and3 unterminated, input 1 terminated

Split bank operation, one MCFD-16 supplies signals to one bank of MQDC-32

0x6040 = 1 // bank_operation → operate banks independently

Experiment gate delay (exp. gate width determines the coincidence time, delay is adjusted here)

0x6054 = 1000 // exp_trig_delay bank0 -> experiment trigger comes 1000ns delayed

0x6056 = 1000 // exp_trig_delay bank1 -> experiment trigger comes 1000ns delayed

Then register setting for FIFO control and readout is required

Register setup MTDC:

The MTDC-32 is operated in connected bank mode. All timings over all modules are differences between the time stamp. So Jitter of the trigger 0 will cancel out as long as the same trigger is fed to all modules. Here trigger1 can be used as an additional timing channel.

IOs

```
0x6062 = 0    //ECL_term -> bit 0 for: "trig0"
              Last module terminates (0x6062 = 1)

0x6068 = 1    // Trigger 0 from ECL3-input
```

Window of interest

```
0x6050 = 0x4000 - 1000    //bank0_win_start.   trigger0 is delayed by 1000ns
                          // so shift window of interest back in time
0x6054 = 500              //bank0_win_width.  500ns window width
```

Trigger source

```
0x6058 = 1          // trigger 0 input triggers the window of interest
```

Data conversion

```
0x6042 = 4          // set resolution to 32ps
0x605C = 0          // only transmit the first hit per channel in the window of interest.
```

Then register setting for FIFO control and readout is required

FIFO Setup, all modules

One TDC can be initialized to signal data via Interrupt:

```
0x6010 = 1  // IRQ = 1, vector = 0 (default)
0x6018 = 1  // signal an interrupt when data detected
```

All modules can then be initialized the same way

```
0x6032 = 0  // event by event transfer
0x603A = 1  // start DAQ (default)
0x603c = 0  // clear FIFO
0x6034 = 0  // readout reset, allow new event
```

Readout loop

```
at IRQ:
        read each module until Berr
        write 0x6034 for next event.
```

Example 3, use the mesytec control bus

Example for controlling external modules with mesytec RC-bus:
Initialize and read out a MCFD16- module.
MCFD16 ID-coder set to 7
Bus line must be terminated at the far end.

Activate MDPP-16 control bus at busy line
Write(16) addr 0x6074 data 1

Get Module ID-Code (= Type of module = 26 for MCFD16)

```
Write(16)    addr 0x6082  data 7      // address module 7
Write(16)    addr 0x6084  data 6      // send code "read IDC"
Write(16)    addr 0x6088  data 0      // initialize send request. Data has no effect
```

Wait loop: Read(16) 0x608A and compare bit0 to get 0. Then evaluate other bits for error status

```
Read(16)     addr 0x6088  data 40     // at ID readout the bit 0
                                         shows the module RC status

                                      // (1 is on).   Bit 1..7 show the IDC
                                      // → interpretation: Module off, IDC = 20
```

Set threshold for channel 0 to 10

```
Write(16)    addr 0x6082  data 7      // address module 7
Write(16)    addr 0x6084  data 16     // code "write_data"
Write(16)    addr 0x6086  data 0      // address module memory location 1
Write(16)    addr 0x6088  data 10     // start send . Data to send
```

Wait loop: Read(16) 0x608A and compare bit0 to get 0. Then evaluate other bits for error status

Optional the read back data is available.

```
Read(16)     addr 0x6088  data 10     // read back written data for control
```

Read threshold of channel 0

```
Write(16)    addr 0x6082  data 7      // address module 7
Write(16)    addr 0x6084  data 18     // code "read_data"
Write(16)    addr 0x6086  data 0      // address module memory location 1
Write(16)    addr 0x6088  data 0      // send read request. Data has no effect
```

Wait loop:
Read(16) 0x608A and compare bit0 to get 0. Then evaluate other bits for error status

```
Read(16)      addr 0x6088  data 10       // read out data,  "10" returned
```

Activate RC in module

All set data will get active. This can also be done before setting the values.

```
Write(16)     addr 0x6082  data 7        // address module 7
Write(16)     addr 0x6084  data 3        // send code "RC_on"
Write(16)     addr 0x6088  data 0        // initialize send request. Data has no effect
```

Deactivate MDPP-16 control bus at busy line

```
Write(16)      addr 0x6074 data 4        // busy output used as busy
```