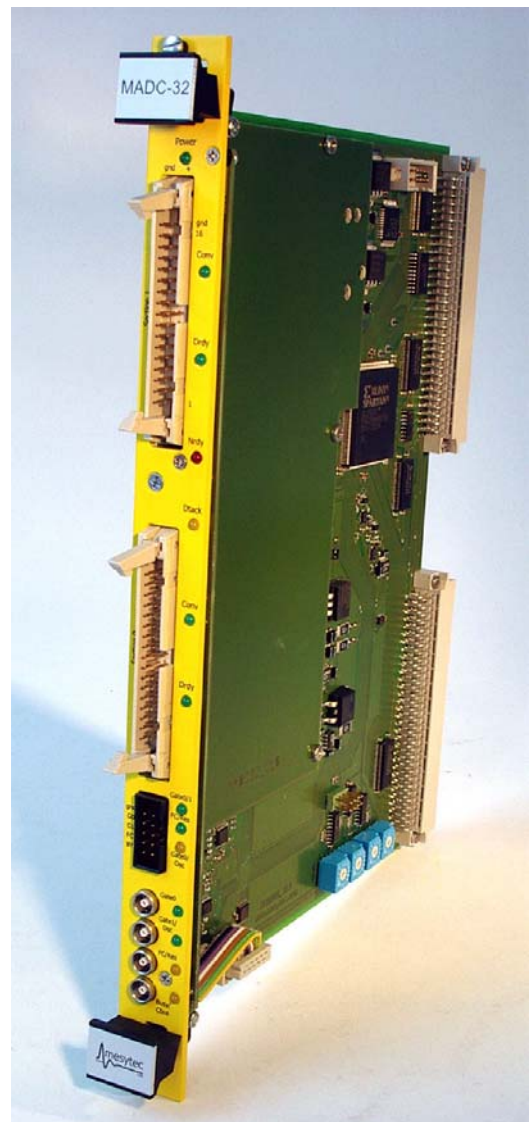


MADC32 is a fast 32 channels peak sensing ADC. It provides a 11 to 13 Bit (2 to 8k) resolution with low differential non linearity due to sliding scale methode.

The conversion time is 800 ns for 32 channels at 2k resolution. It supports zero suppression with individual thresholds.

## Features:

- High quality 11 to 13 bit (2, 4, 8 k) conversion with sliding scale ADC (DNL < 1% @ 4k).
- 800 ns, 1.6us, 6.4us conversion time for 32 channels wit 2k, 4k, 8k resolution.
- 1 k words multi event buffer
- Zero supression with individual thresholds
- Supports different types of time stamping
- Independent bank operation
- Two register adjustable gate generators are built in
- Input range, register selectable 4V, 8V,10V
- Mesytec control bus to control external mesytec modules
- Address modes: A24 / A32
- Data transfer modes: D16 (registers), D32, BLT32, MBLT64, CBLT, CMBLT64
- Multicast for event reset and timestamping start



## MADC-32 data

### Input / Output

Conversion input 1 kOhm, 4 V, 8 V or 10 V configurable via register  
 Risetime min: 50 ns, max: DC-conversion possible

ECL inputs:  
 standard ECL input, can be individually terminated via register setting

NIM inputs:  
 standard NIM

NIM output:  
 -0.7 V terminated

mesytec control bus output, shares connector with busy output. +0.7V terminated

### Digital Inputs /outputs (see IO register block 0x6060)

Input /output	direction	termi-nation	Default func-tionality	Alternate functionalities
<b>ECL0</b>	Input	R*	Gate0	-
<b>ECL1</b>	Input	R	Gate1	Time stamp oscillator input
<b>ECL2</b>	Input	R	Fast clear	Reset time stamp counter
<b>ECL3</b>	Output	100R	Busy	-
<b>NIM0</b>	Input	50R	Gate0	-
<b>NIM1</b>	Input	50R	Gate1	Time stamp oscillator input
<b>NIM2</b>	Input	50R	Fast clear	Reset time stamp counter
<b>NIM3</b>	Input / Output	50R	Busy	2) Mesytec control bus I/O  for monitoring and adjustment: 3) internal gate generator 0 output 4) internal gate generator 1 output

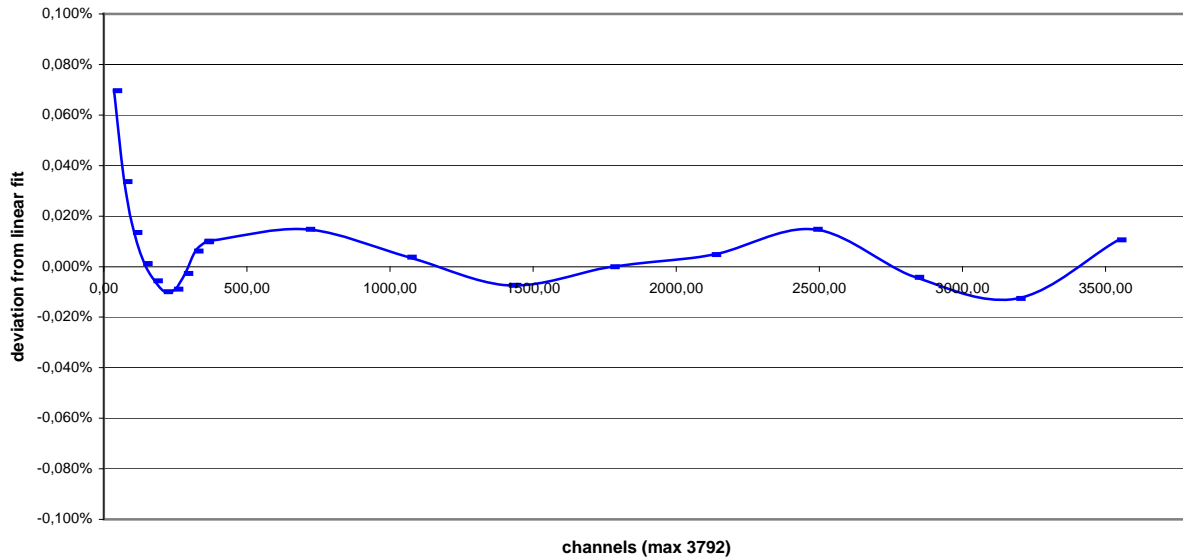
“R” means register selectable termination.

### Front Panel LEDs:

- LED “Conv” digitization in progress
- LED “Drdy” Data are ready converted and can be read out
- LED “Nrdy” Gate detected, but ADC busy. Event will be lost (should never happen in a well controlled DAQ). Or fast clear outside of acceptance window.
- LED “Dtack” Acces from VME bus accepted

**Integral non linearity**

INL <  $2 \cdot 10^{-4}$  in the range 2% to 100% (@ 2us shaping time, 4k-mode 3.2us conv time).

**MADC32, INL**

**Output formats, resolution, conversion time**

The sliding scale needs 1/16 of the full binary value.

Conversion	Typical Noise in channels rms	Max noise Channel rms	Conversion time 32 channels	highest channel / overflow channel
2k	0.6	0.8	0.8us	1919 / 1920
4k	0.9	1.2	1.6us	3839 / 3840
4k hires	0.7	1.0	3.2us	3839 / 3840
8k	0.9	1.2	6.4us	7679 / 7680
8k hires	0.7	1.0	12.8us	7679 / 7680

**Power consumption (Total: 4.5 W)**

- +5 V, +230 mA
- +12 V, +160 mA
- 12 V, -100 mA

**Conversion, busy time**

1. Digital conversion time:
2. 800 ns for 32 channels @ 2k resolution
3. Conversion starts 50 ns after gates closes
4. Recovery time after conversion: 200 ns

**Lemo and ECL inputs**

Minimum gate width: 100ns

Minimum clear signals: 50ns

Maximum external reference oscillator frequency: 70MHz

**Gate generators**

Two gate generators are provided for each bank of 16 channels.

When the gate generators are used, the gate inputs work as trigger inputs to start the gate generators.

When active the gate generators create a gate for the stretchers. Delays and widths can be adjusted independently in steps of 50 ns. Minimum delay (gate\_delay parameter = 0) is 25ns.

For monitoring, the gate signals can be switched to the busy output via register setting (reg. 0x606E).

## MADC32 register set.

### Data FIFO, read data at address 0x0000 (access R/W D32, 64)

for 64 bit access, last 32 bits will be filled with 0 for odd number of 32 bit data words  
 memory size: 1024 + 2 = 1026 words with 32 bit length

#### Header (4 byte)

2 header sig- nature	6 sub- header	8 module id	1 output_format -> 0x6044	3 adc_resolution -> 0x6042	12 number of following data words, including EOE
b01	b000000	module id	bx	bxxx	number of 32 bit data words

#### Data (4 byte) DATA event

2 data-sig	9	5	1	1 out of range	1..3	11..13
b00	00 0100 000	channel number	b0	Oor	b00	ADC amplitude

channel numbers may come in arbitrary order

#### Data (4 byte) Extended timestamp

2 data-sig	9	5	16			
b00	00 0100 100	0 0000	16 high bits of timestamp			

#### Data (4 byte), fill dummy (to fill MBLT64 word at odd data number)

2 data-sig	9	5	1	1 -	2	12
b00	0	0	0	0	0	0

#### End of Event mark (4 byte)

2	30
b11	trigger counter / time stamp

### Threshold memory at address x4000 to x403F (16 bit words, access: R/W D16)

Address	Name	Bits	dir	Default	Comment
0x4000	treshold[0]	13	RW	0	Threshold value of channel 0 value 0 = threshold not used
.					
.					
.					
0x403E	treshold[31]	13	RW	0	Threshold value channel 31

Threshold values are referenced to the actually adjusted data output width (2k, 4k or 8k). Reg 0x6044, 0x6046. With the value 0x1FFF the channels are switched off in any resolution mode.

**Registers, Starting at address x6000 (access D16)**

Address	Name	Bits	dir	default	Comment
	<b>Address registers</b>				
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 (board coder).
0x6008	soft_reset	1	W		breaks all activities, sets critical parameters to default
0x600E	firmware_revision	16	R		0x01.10

	<b>IRQ (ROACK)</b>				
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)
0x6016	irq_reset	0	W		resets IRQ (for test)
0x6018	irq_threshold	10	RW	1	Every time the number of 32bit words in the FIFO exceeds this threshold, an IRQ is emitted. Maximum allowed threshold is 956.
0x601A	Max_transfer_data	11	RW	1	Maximum data words to transfer before ending the transfer at next end of event word. Only works for multievent mode 3. At Max_transfer_data = 1, 1 event per transfer is emitted. Maximum number of events is 2047. Usually the same or higher value than in 0x6018 is used. Setting the value to 0 allows unlimited transfer.
0x601C	Withdraw IRQ	1	RW	1	Withdraw IRQ when data empty

For multievent mode 2 and 3 the IRQ is:

**-set** when the fifo fill level gets more than the threshold and is

**-withdrawn** when IRQ is acknowledged or when the fill level goes below the threshold.

	<b>MCST CBLT</b>				
0x6020	cbtlt_mcst_control	8	RW	0	see table
0x6022	cbtlt_address	8	RW	0xAA	A31..A25 CBLT- address
0x6024	mcst_address	8	R	0xBB	A31..A25 MCST- address

0x6020: CBLT\_MCST\_Control

Bit	Name	Write	Read
7	MCSTENB	1 Enable MCST 0 No effect	0
6	MCSTDIS	1 Disable MCST 0 No effect	1 MCST enabled 0 MCST disabled
5	FIRSTENB	1 Enable first module in a CBLT chain 0 No effect	0
4	FIRSTDIS	1 Disable first module in a CBLT chain 0 No effect	1 First module in a CBLT chain 0 Not first module in a CBLT chain
3	LASTENB	1 Enable last module in a CBLT chain 0 No effect	0
2	LASTDIS	1 Disable last module in a CBLT chain 0 No effect	1 Last module in a CBLT chain 0 Not last module in a CBLT chain
1	CBLTENB	1 Enable CBLT 0 No effect	0
0	CBLTDIS	1 Disable CBLT 0 No effect	1 CBLT enabled 0 CBLT disabled

**CBLT Address Field**

A31 ..... A24	A23..... A00
CBLT ADRS	8 high bits, not significant + 16 bit module address space

**MCST Address Field**

A31 ..... A24	A23..... A00
MCST ADRS	8 high bits, not significant + 16 bit module address space

**At BLT32 :**

When an empty module is accessed at address 0, BERR is emitted.

**At CBLT:** when no module contains data, no data are transmitted. The last module emits BERR. Usually when zero suppression is used and all modules were gated, each Module emits the header and footer with time stamp (2 Words with 32 bits each: MAD32 Header, MAD32 footer).

<b>FIFO handling</b>					
0x6030	buffer_data_length	14	R		amount of data in FIFO (only fully converted events). Units -> data_len_format. Can be used for single- and multi event transfer
0x6032	data_len_format	2	RW	2	0= 8 bit, 1=16 bit, 2=32 bit, 3=64 bit At 3 a fill word may be added to the buffer to get even number of 32 bit words.
0x6034	readout_reset		W		At single event mode (multievent = 0): allow new trigger, allow IRQ At multievent = 1: checks threshold, sets IRQ when enough data. Allows safe operation when buffer fill level does not go below the data threshold at readout. At multievent = 3 : clears Berr, allows next readout
0x6036	multievent	4	RW	0	allow multi event buffering (bit 0,1) <b>0=no</b> (0x6034 clears event, allows new conversion) <b>1= yes</b> , unlimited transfer, no readout reset required (0x6034 can be written after block readout) . Don't use for CBLT <b>3= yes</b> but MADC transfers limited amount of data. With reg 0x601A the number of data words can be specified. After word limit is reached, the next end of event mark terminates transfer by emitting Berr. So 0x601A = 1 means event by event transfer (Berr after each event). The next data block can be transferred after writing 0x6034 (resets Berr).  <b>Berr handling:</b> when bit 2 is set: Send EOB = bit[31:30] = bx10 instead of Berr  Bit 3: Compare number of transmitted events with max_transfer_data (0x601A) for Berr condition.
0x6038	marking_type	2	RW	0	00 -> event counter 01 -> time stamp 11 -> extended time stamp (only at banks connected)
0x603A	start_acq	1	RW	1	1 -> start accepting gates If no external trigger logic, which stops the gates when daq is not running, is implemented , this register should be set to 0

					before applying the fifo_reset to get a well defined status. When setting it to 1 again for data acquisition start, the buffer is in a well defined status.
0x603C	fifo_reset		W		initialise fifo
0x603E	data_ready	1	R		1 -> data available

<b>operation mode</b>					
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 -> 2k (800ns conversion time) 1 -> 4k (1.6us conversion time) 2 -> 4k hires (3.2us conversion time) 3 -> 8k (6.4us conversion time) 4 -> 8k hires (12.5us conv. Time)
0x6044	output_format	1	RW	0	0 -> addressed mesytec format.
0x6046	adc_override	2	RW	2	When written, it overrides the channel output width (2k...8k) but not the conversion time. Values defined as in "adc_resolution"
0x6048	slc_off	1	RW	0	Switch off sliding scale
0x604A	skip_oorange	1	RW	0	Skip out of range values

<b>gate generator</b>					
0x6050	hold_delay0	8	RW	20	0= 25ns, 1= 150ns, 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)

<b>IO</b>	<b>Inputs, outputs</b>				
<b>0x6060</b>					
0x6060	input_range	2	RW	0	input range: 0-> 4V, 1-> 10V, 2-> 8V
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 ( <b>also set 0x6096 !!</b> )
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_gate1_osc	1	RW	0	0 -> gate1 input, 1-> oscillator input ( <b>also set 0x6096 !!</b> )
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	b0000 -> as busy (in independent bank operation or toggle mode: active when both banks are busy) b0001-> as gate0 output b0010 -> as gate1 output b0011 -> as Cbus output b0100 -> buffer full b1000 -> data in buffer above threshold 0x6018
<b>0x6070</b>	<b>Testpulser</b>				
0x6070	pulser_status;	4	RW	0	b000 = off, b100 = amplitude =0 (input-> 10V range) b101 = low amplitude (7%), b110 = high amplitude (75%) b111 = amplitude: 0 -> low -> high ->0...

**Mesytec control bus:**

*Set 0x606E to 3 before using Cbus. Wait 100us to allow updating of output configuration*

<b>MRC 0x6080</b>	<b>Module RC</b>				
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 hexcoder 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=any collision (no termination?) bit3=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

The Gate0-LED shows data traffic on the bus , the Gate1-LED shows bus errors (i.e. non terminated lines)

**Example for controlling external modules with mesytec RC-bus:**

Initialise and read out a MSCF16 Shaper module.  
MSCF16 ID-coder set to 7  
Bus line must be terminated at the far end.

**Activate MAD32 control bus** at busy line  
Write(16) addr 0x606E data 3

**Get Module ID-Code** (=Type of module = 20 for MSCF16)  
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 // initialise 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 gain for channel 1 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 1  // adress 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 contol
```

### Read gain of channel 1

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

### Deactivate MAD32 control bus at busy line

```
Write(16)   addr 0x606E data 0  // busy output used as busy
```

**CTRA**

Timestamp 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. Fast cleared events are not counted.

<b>CTRA</b>	<b>countersA</b>				
<b>0x6090</b>					
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
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	timestamp = 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

For counters "ADC\_busy" and "Gate1\_busy" the count basis is 25ns.

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

<b>CTRB</b>	<b>countersB</b>				
<b>0x60A0</b>					
0x60A0	adc_busy_time_lo	16	R		ADC busy time, from gate to end of conversion. Step [1us]
0x60A2	adc_busy_time_hi	16	R		
0x60A4	gate1_time_lo	16	R		Gated time from Lemo gate 1 input [1us] timer counts when gate1 has active NIM level (-0.6V). Step [1us]
0x60A6	gate1_time_hi	16	R		
0x60A8	time_0	16	R		time [1us] (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)

## Data handling

The event buffer is organised as a FIFO with a depth of 1 k x 32 bit.

Data is organized in an event structure, maximum size of one event is 34 32-bit words.

### Event structure:

Word # (32 bit)	Content
0	Event header (indicates # of n following 32-bit words)
1	Data word #1
2	Data word #2
...	...
n-1	Data word #n-1
n	End of event marker

### Event Header (4 byte, 32 bit)

Short #1																Short #0																	
Byte #3								Byte #2								Byte #1								Byte #0									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
hsig		subheader						module id								f	adc res				# of following words												
0	1	0	0	0	0	0	0	i	i	i	i	i	i	i	i	f	r	r	r	r	n	n	n	n	n	n	n	n	n	n	n	n	n

hsig: header signature = b01

subheader: currently = b000000 → Byte #3 = 0x40

module id: depending on board coder settings → Byte #2 = Module ID

output format f: currently = 0 (addressed format)

adc res: ADC resolution, depending on register 0x6044  
 0 = 2k, 800 ns conv. Time  
 1 = 4k, 1.6 us conv. Time  
 2 = 4k hires, 3.2 us conv. Time  
 3 = 8k, 1.6 us conv. Time  
 4 = 8k, hires, 3.2 us conv. Time

# of follow. words: indicates amount n of following 32-bit words:  
 n-1 events + 1 end of event marker)

**Data words (4 byte, 32 bit) DATA-event**

Short #1																Short #0																	
Byte #3								Byte #2								Byte #1								Byte #0									
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
dsig		fix								channel #								V	ADC data (11-13 valid bits)														
0	0	0	0	0	1	0	0	0	0	0	0	c	c	c	c	c	0	v	0	d	d	d	d	d	d	d	d	d	d	d	d	d	d

dsig: data signature = b00

fix: bitfield currently without meaning = b000100000 → Byte #3 = 0x04

channel #: number of ADC channel → Byte #2 = channel#  
within an event buffer, ADC channels may occur in arbitrary order

V: V=1 indicates ADC overflow

ADC data: ADC conversion data, data width 11...13 valid bits, depending on register 0x6044

**End of Event mark (4 byte, 32 bit)**

Short #1																Short #0															
Byte #3								Byte #2								Byte #1								Byte #0							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
esig		trigger counter / time stamp (30 bit)																													
1	1	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t

esig: end of event signature = b11

trigger counter/  
time stamp 30 bit trigger counter or time stamp information, depending on register 0x6038 “marking type”: 0 = event counter, 1 = time stamp

When in single event mode (register 0x6036 = 0), reading beyond EOE, MADC-32 emits a VME Berr (bus error).

When in multi event mode 3 (register 0x6036 = 3), reading beyond EOE after the limit specified in register 0x601A, MADC-32 emits a VME Berr (bus error)

This can be used to terminate a block transfer or multi block transfer.

## The MAD32 read out in two modes:

### Single event readout

In this mode the pulses are stretched and converted starting with an external gate or trigger. The data are then stored in a memory and the module waits for the VME readout. After readout of the data at 0x0000 the register 0x6034 is written and allows a new gate to start the conversion. Gates coming within the time from first gate to writing the 0x6034 register are ignored. For dead time the conversion time and VME readout time add up.

1) Assumed: 32 bit read (D32 or BLT32)

Wait for IRQ to start readout of an event

Read register #6030 for event length

Read from buffer event\_length + 1

Write reset register 0x6034

2) After IRQ start block transfer until BERR on VME-bus

Then write reset register 0x6034

### Example

Stop acquisition: start\_acq 0x603A = 0; Stop

Set multievent register 0x6036 = 0 (default).

At power up reset or after soft reset, the IRQ register is set to 0 (no interrupt)

Initialise IRQ (for example to IRQ1, Vector = 0):

set IRQ:

set reg 0x6012 to 0 (IRQ Vector)

set reg 0x6010 to 1 (IRQ-1 will be set when event is converted)

Reset fifo: write register 0x6034 (any value)

start\_acq: 0x603A = 1; Start

Now module is ready for IRQ triggered readout loop:

-> IRQ

Read register 0x6030 for event length (D16)

Read from buffer event\_length + 1 (BLT32)

Write reset register 0x6034 (D16)

Or:

-> IRQ

Start block transfer (BLT32) until BERR on VME-bus

Then write reset register 0x6034 (D16)

The above procedure works completely unchanged **with multi event mode 0x6036 = 3 and 0x601A = 0**. In this mode the buffer is used but the data are read out event by event. After each event a Berr is emitted, which is removed by writing the 0x6034 readout reset.

## Multi event readout

In multievent readout mode (0x6036, mutievent = 1 or 3) the input is decoupled from output by a 1024 words buffer. So the input is ready for a new gate after the conversion time of the ADC.

When several converter modules are used in one setup, there has to be a way to identify coincident data from different modules which belong to the same event.

## Event synchronisation

One method is **event counting**.

Each module has an event counter and counts the incoming gates. In complex setups, the gates are best initiated by the individual detector timing signals and significant amount of logic and timing modules have to be established and adjusted to coordinate the detector triggers. Also fast clears may be necessary for some detectors. A single timing error in all the experiment run time, which will allow an additional gate to come to some module or a suppression of a gate, will corrupt the complete data set, as data get asynchronous.

The better one is **time stamping**.

A central oscillator clock (for MAD32 this can be the VME built in clock of 16MHz or an external clock up to 70MHz) is counted to create a time basis. At experiment start the time counters of all modules are reset via a VME multicast write to a reset register, or by an external reset signal. All incoming events are then labeled with a 29 bit long time tag. At data analysis the data streams from different modules are analysed and correlated events are grouped for further processing.

The synchronisation methods allow the different modules to be completely independent from each other. It gets now possible to use large data buffers in the frontend modules, and do the readout when the VME data bus is not occupied. The MAD32 allows to set a buffer fill threshold which emits an interrupt when the data fill level in the buffer exceeds the threshold.

## Data transfer:

In principle any amount of data can be read at any time from the buffer, but then events may be split to two consecutive readout cycles, which normally is no problem.

When only full events should be read in one readout cycle, there are two possibilities.

1) multievent mode = 1: read "buffer\_data\_length" (0x6030) and transfer the amount of data read there.

2) multievent mode = 1: The buffer must be read to the end which means to the Berr mark. Note that this in principle requires to read an infinite number of words, because at fastest conversion the dead time may be as low as 1µs, the amount of data without zero suppression may be 34 words per conversion. So the theoretical amount of data written to the buffer can be up to 34 Mwords/s, the VME readout rate is realistic about 5 Mwords /s in BLT32. So under worst conditions it is not possible to empty the buffer via VME and get an empty fifo signal "Berr" !

So if high rates can appear, the data acquisition should at least be tolerant to splitted events.

3) an easier way to overcome those problems is to use multievent mode = 3 and limit the data transfer via register 0x601A to a reasonable amount (for example 1000 Words). After readout, 0x6034 has to be written to allow transmission of a new data block.

## IRQ

For many setups it is useful to control the readout via interrupt requests (IRQ) defined by VME. For MADC32 an IRQ is initiated when the buffer fill level gets above the “irq\_threshold” (0x6018). The IRQ is acknowledged by the VME controller, then the controller starts a readout sequence. When not using the readout reset (0x6034) at the end of a readout cycle, the MADC does not know when the cycle ends. The IRQ is then set again when the data fill level exceeds the irq-threshold. When not enough data are read from fifo to drive the fifo fill level below the threshold, no new IRQ will be emitted.

So for a readout which is stable against any external influences (readout delays, high input rates), we recommend to write the readout\_reset after each readout sequence. For several MADC modules in VME bin this can also be done with a single multicast write.

## Example 1, multievent readout

### 1) Stop acquisition

start\_acq 0x603A = 0; Stop

### 2) Time stamping

The module will use here an external reference oscillator and will be reset (synchronised) via VME command.

Set oscillator input	ECL_gate1_osc	0x6064 = 1;
Set oscillator source, reset source	ts_sources	0x6096 = 2; (ext osc, int reset only)
show timestamp in EOE mark	marking type	0x6038 = 1;
Synchronisation:	Reset_ctr_ab	0x6090 = 3; reset all counters

### 3) Set Multievent

Multievent 0x6036 = 3                      multievent with limited data transfer  
Irq\_threshold 0x6018 = 200              Irq is set when more than 200 (32 bit-)words are in buffer  
Max\_transfer\_dat 0x601A = 218 transmit maximum 223 words + rest of event before sending Berr.  
(In this case data fits into one VME 255 word blt32 transfer)

### 4) IRQ

Initialise IRQ (for example to IRQ1, Vector = 0):

set IRQ:

set reg 0x6012 to 0 (IRQ Vector)  
set reg 0x6010 to 1 (IRQ-1 will be set when event is converted)  
set reg 0x6018 to 100 (IRQ emitted when more than 100 words in fifo)

### 5) Buffer initialisation, start

Fifo\_reset 0x603C = 0;  
Readout reset 0x6034 = 0;  
start\_acq 0x603A = 1; Start

## 6) Readout loop

-> IRQ

Start multi block transfer (BLT32 or MBLT64) until BERR on VME-bus  
Then write reset register 0x6034 (D16)

## Example 2, chained block transfer

Describes multi event readout but with 3 MADCs and chained block transfer

To operate several modules in one VME bin, each module has to be given a different address. The 4 coders on the main board code for the highest 16 bits of the 32 bit address. Best way is, to use only the highest 8 bits for coding (2 rotary coder marked with high). It makes sense to use the slot number as high address. So:

ADC1 in slot 1 gets 0x0100

ADC2 in slot 2 gets 0x0200

ADC3 in slot 3 gets 0x0300

If you don't change the module ID default, the modules will now also have the ID 1...3 which will be transmitted in the data header.

Now initialise the individual modules:

ADC1: set 0x0100 6020 to 0xA2 (CBLT first module, Multicast enable)

ADC2: set 0x0200 6020 to 0x82 (CBLT mid module, Multicast enable) also any further module in the middle of the readout chain is initialised this way.

ADC3: set 0x0300 6020 to 0x8A (CBLT last module, Multicast enable)

When you don't change the default addresses for CBLT and MCST, the modules will have the CBLT start address of 0xAA00 0000 and the MCST start address of 0xBB00 0000.

You can now do the initialisation 1) to 5) of Example 1 via multicast at the offset address 0xBB00.

The readout loop has to be modified slightly:

-> IRQ

Start multi block transfer (BLT32, MBLT64) at address 0xAA00 0000 until BERR on VME-bus

Then write reset register 0xBB00 6034 (D16) at the multicast address.

**Note:** use multievent mode 0 or 3 for CBLT (mode 1 will not work !)

## Special Operation

### MBLT64,

MBLT64 is defined by the address modifier. At buffer unpacking in analysis software it helps to keep the alignment within the transmitted 64 bit word. When setting register 0x6032 to 3 a fill word is added to the converted channels when converted channel number is odd.

### CMBLT64

Is intrinsic when chained block transfer is used with MBLT64.

**Note:** Register 0x6032 has to be set to 3 to align the words !

### Using the built in gate generators

The MAD32 provides built in gate generators (for joined bank operation only gate generator0 is used). Only a short trigger has to be delivered to the gate input to start the gate generator. Delay and width of the gate can be set via registers.

To use the gate generators, they have to be defined as gate source in 0x6058. For monitoring of the generated gate, it is available at the busy output when setting 0x606E to 1 (gate 0) or 2 (gate1).

### Using the two banks independently

The MAD32 can work as two independent 16 channel ADCs. In this mode it creates independent event structures for the two banks while the 5 bit (0..31) channel numbers are kept in the data words.

### Using the toggle mode for 16 channels (very low dead time)

The data source (Shaper output) has to be connected to both input banks in parallel.

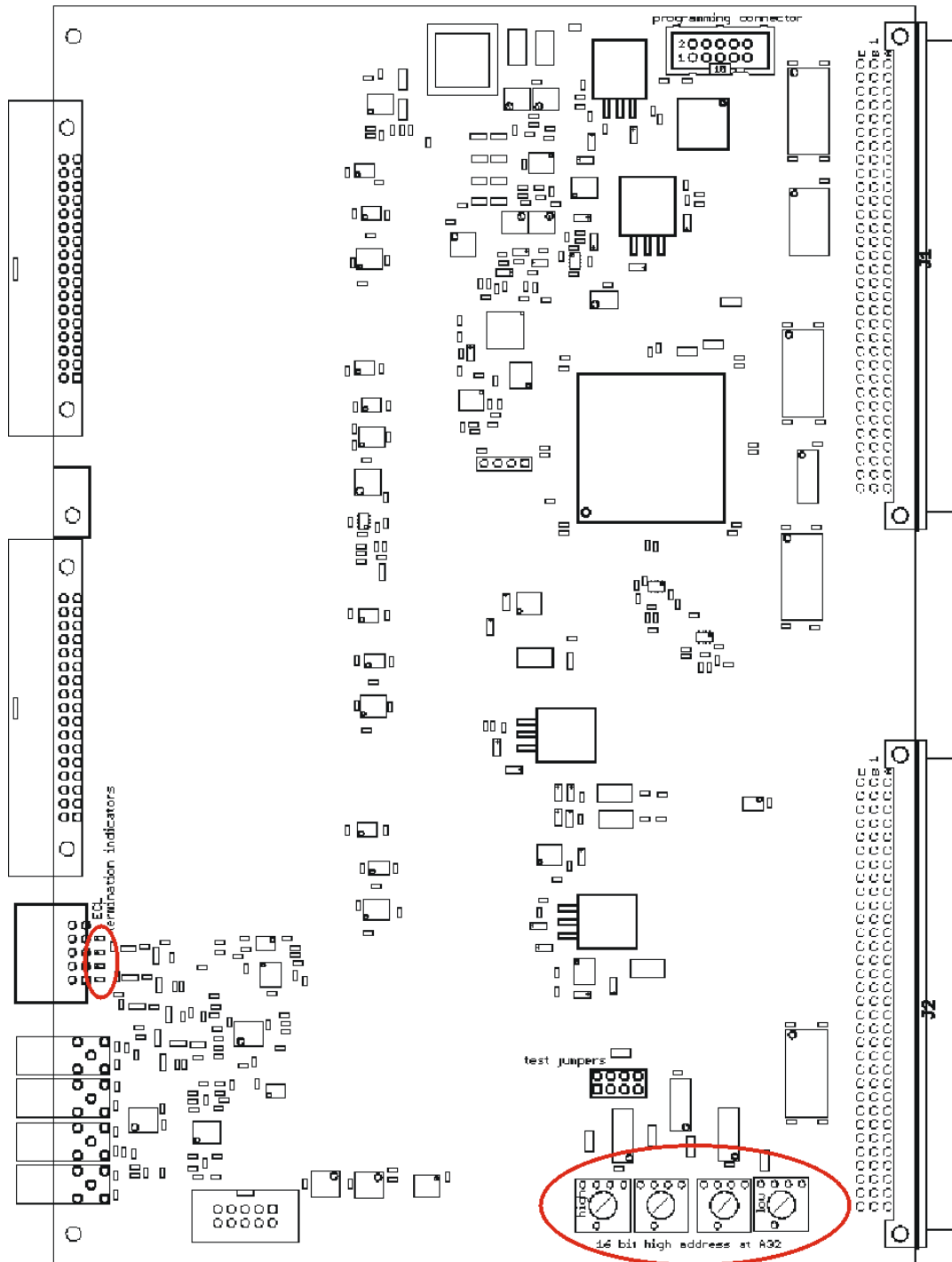
The gate0 input is used as trigger input, which means only a short pulse per conversion is needed. The trigger pulses alternately start one of the two gate generators with their associated bank. So while one gate generator or conversion is active in one bank, the other is ready for the next trigger. The draw back is the reduction of channel number to only 16 for an MAD32 module.

### Fast clear

For some application it is useful to reject an event with some delay needed for trigger decision. The MAD32 allows to clear an event from gate start to end of conversion (this can be 800ns to 12.5us, depending on conversion time setting). Also when the built in gate generators are used the fast clear signal stops and resets them immediately. Fast clear time (from active edge of the fast clear signal to next gate) is about 200ns. The fast clear signal must be withdrawn before next gate gets active.

**MADC32**

Address coders,  
programming connector  
test jumper position  
3x ECL termination indicators for ECL inputs.



## Changes

- FW01.19 - IRQ register 0x601C implemented, default 1. When set to 0 strict IRQ handling is active -> is not withdrawn when fifo empty.
- FW01.21 - VME sysclk deglitch to allow synchronous time stamping,
  - busy output : also buffer full and buffer threshold possible (0x606E)
- FW 0122 - "channel crosstalk" with internal gate fixed
  - added event number limit when multi event mode 0x6036 = 4'b1011 is set
  - extended timestamp can be added to data stream (reg 0x6038)
  - latch of time stamp shifted to beginning of gate.
  - FW0122 needs offset recalibration: remove all front connectors, set 0x6100 = 0x1234, wait some ms, set 0x610E = 0, wait 30 seconds, power cycle. NRDY LED light up during calibration indicates error status.
- FW0123 - CBLT, BLT32, MBLT64 read access is always directed to the FIFO address 0 (regardless of the least significant 16 bit address)
- FW0124 - extended time stamp reset implemented
- FW0125 - improved VME bus filtering, decreases error rate at CMBLT64 and all other VMEbus operations
- FW0126 - Single shot reset for time stamp implemented (synchronisation with Xia modules)