NAME

TRLO II - trigger logics - in VULOM and TRIDI

DESCRIPTION

The VULOM-based trigger logics replaces several crates of NIM and CAMAC electronics used for trigger decisions, counting and dead-time locking --- and open up many new possibilities for handling pending and calibration triggers. As the TRIDI modules need similar kinds of logics, and actually could serve as small local trigger systems for stand-alone test operations, the same code is used for both, with minor tweaks for the different in- and outputs. All logics run with one 100 MHz clock.

The bold intention is that this firmware should be suitable for (almost) any triggered nuclear physics experiment. If you can think of something useful which it cannot do, but that would make it useful also for you, do not hesitate to contact the author --- customisations handled by general extensions are of interest.

The layout of the address space of the VME registers is dependent on the configuration of the VHDL code, i.e. the number of various resources implemented. To make use easy, the location of all registers are calculated at compile time, and corresponding C structures that can be used for addressing are generated as trlo_big_defs.h. (The use of a base pointer and offsets is anyhow not an approach recommended by the author.) The version of the code is identified by a md5sum of the entire VHDL source, to avoid incompatibilities between versions.

OVERALL LAYOUT

The TRLO_BIG firmware consists of several parts:

o 'Fast-path' for input detector-signals to master start:

- Variable delay (trigger alignment).
- Self-triggering soft-scope to help with trigger alignment.
- Stretcher.
- Logic matrix (LMU) (coincidence building), with aux inputs.
- Dead-time veto.
- Reduction (downscale).
- Sum-out generation (master start).
- Scalers (each trigger, before LMU, before & after DT, after red).

o Trigger state machine, the data acquisition cycle control:

- Idle.
- Coincidence acceptance window.
- Mapping TPAT to TRIVA triggers.
- Priority selection and encoding.
- Send encoded trigger to TRIVA.
- Multi-event mode by sending no TRIVA trigger, but wait for busy release of converter modules.
- Limit on the number of multi-events.
- Multi-event TPAT (and event-time) buffer.

- Wait for dead-time release.
- Handling of pending (guaranteed) triggers.

o A large array of signal multiplexers. To combine the functionality of the trigger logics itself and the auxiliary logics, a unified approach where all signal sources, i.e. the VULOM inputs as well as function generator outputs are treated on an equal footing. The same applies to all VULOM outputs, and inputs to function generators and monitoring as well as the auxiliary sources for the trigger logic. The only exception is the fast-path itself, which for latency reasons is directly connected.

Sources:

- VULOM/TRIDI inputs.
- Pulsers.
- General logic functions (gate & delay, LMU, edge-to-gate, etc.).
- Accepted triggers.

Destinations:

- VULOM/TRIDI outputs.
- LEDs, scalers and latches, soft-scope.
- Stimuli for logic functions.
- Trigger control, including external dead-time.

Any destination can use any source.

There is also the possibility to connect any module output to receive from any input without clocking, i.e. without clocking jitter.

o General logic functions:

- Pulsers (programmable frequency).
- PRNG (pseudo-random sequence).
- LMU (not the same as in fast-path).
- Downscale.
- Delay and stretch (a.k.a. gate-and-delay).
- Edge-to-gate conversion (e.g. spill mimic).
- Fan-in (masked all-or).
- Coincidence.

o Monitoring:

- Scalers. Latched with selectable input.
- Timer latches; latch the value of a global time counter on a signal edge.

- Multi-entry buffer for the timer-latches.
- Self-triggering soft-scope.
- Input pattern latch.
- Front-panel LEDs.
- Front-panel display.

FAST-PATH

This replaces the previous chain of alignment delays, LMUs and trigger boxes.

For each detector signal, each connected to one of the 16 VULOM ECL inputs (or 8 ECL and 8 NIM TRIDI inputs) the signal is clocked. In

In the figure below, ... denotes clock boundaries, i.e. FPGA pipe-line stages. Note that all diverted signals not are time critical and therefore separated by clocking at the diversion.

```
   ECL-INPUT          (1-16)
 ..... | .....
  DELAY-LINE          (alignment, shift register)
     |
     +--> SOFTSCOPE  (for alignment)
     |
   STRETCHER          (countdown, restartable)
     |
     +--> SCALER     (before LMU, own l.e. detect)
     |
+-- LMU INPUT --+
|            |---< LMU AUX IN (from general logics)
+- LMU OUTPUTS -+
  ..... | .....
     +--> SCALER     (after LMU / before dead-time)
     |
     +--> OR         (to avoid inhibit release at trailing trigger)
     |
 DEADTIME VETO       (inhibit signal from trigger state machine)
     |
  L.E. DETECT        (make single-cycle pulses)
     |
     +--> SCALER     (after dead-time)
     |
   REDUCTION         (counter + mux of selected factor 2^n)
     |
     +--> SCALER     (after reduction)
     |
     +--> TPAT       (to trigger state machine)
     |
     OR              (any accepted trigger makes an accepted event)
     |
   STRETCHER         (countdown)
 ..... | .....
  MASTER START       (stretched, selectable output (via mask))
```

Note that the number of LMU outputs = number of TPAT bits need not be the same as the number of LMU

inputs.

Note that any signal that make it past the dead-time veto, i.e. possibly to the master start (provided downscale is right) *is* an accepted trigger. This is arranged together with the trigger state machine by it letting enough cycles pass to ensure that all such signals are allowed to combine into the recorded TPAT. It is also carefully respected in case of internal (pending) triggers.

By applying the leading-edge detect (which turns long multi-clock signals into single-clock pulses) before the scaler before DT, all remaining logics, in particular scalers and the reduction counter, can work directly on the propagated information.

By running the fast-path at a relatively slow clock (100 MHz) instead of something faster it is possible to combine the major part of the operations into two pipe-line stages, instead of many more which then also would have incurred more overhead for each clock cycle (flip-flop sample-and-hold) and be more uneven in the usage of the cycles.

LOGIC MATRIX SETUP

The logix matrix is modelled after the LeCroy 2365 CAMAC module. Each output j is independent. For each output, it can be selected which inputs shall affect the result and how, according to:

```
out(j) = reg_not(j) XOR
        OR_OVER_ALL_i ( (    reg_and(i,j) AND     in(i)) |
                    (reg_not_and(i,j) AND NOT in(i)) )
```

The setup are set via the control registers trig_lmu_not (reg_not) and trig_lmu_[n]and (reg_and and reg_not_and). The above formula can produce a simple OR function by setting reg_not and reg_not_and to 0 and let reg_and select which inputs to include. To achieve the functionality of an coincidence/anti-coincidence unit (as is usually wanted for triggering), set reg_not to 1, reg_not_and to 1 for all signals required in the coincidence and reg_and to 1 for all signals that shall be absent (veto). This achieves the desired function by generating a 1 from the OR_OVER_ALL_i whenever some signal is in a state that shall prevent the final output, i.e. either a veto is present, or a required coincidence is missing. Input signals having 0 for both reg_not_and and reg_and are ignored (no-care). Setting both reg_not_and and reg_and is a useless setting. Unused outputs must have reg_not set to 0, since otherwise, if no input is selected by either reg_and or reg_not_and, it will give an continuous output signal.

CONTROL REGISTERS

Below, i loops over the LMU inputs, j over the LMU outputs, k over the mux inputs and l over the aux inputs.

trig_delay[i]

The distance between the read and write pointers of the delay RAM to multiplex into the stretcher. Due to implementation details, the minimum value 0 corresponds to a delay of 3 cycles.

trig_delay_mode[i]

Use the direct input instead of the delay-line output (**TRLO_BIG_TRIG_DELAY_MODE**_*xxx*):

DELAY_ZERO - 0 cycles delay
DELAY_ONE - 1 cycle delay
DELAY_TWO - 2 cycle delay
DELAY_LINE - delay by delay-line (add 3)
TEST_INPUT - use the mux control signal test input.

When two copies are needed with different delays, the previous input can be used before the delay instead of the current (**TRLO_BIG_TRIG_INPUT_MODE_***xxx*):

THIS - input [i]
PREV - input [i-1] (wraps at i = 0 to last input)

It is selectable if the stretcher only restarts at the leading edge, or as long as the pulse is active (**TRLO_BIG_TRIG_RESTART_MODE_***xxx*):

LEADING_EDGE - pulse length as set below
WHEN_PRESENT - pulse length prolonged as below

trig_stretch[i]
Length of stretched signal. Should be longer than the coincidence/trigger acceptance window.

trig_mux[k]
Select module input for multiplexed trigger input, i.e. one of **TRLO_BIG_MUX_SRC_***xxx* that marks a real module input.

trig_mux_delay[k]
Analogous to trig_delay for the multiplexed inputs.

trig_mux_delay_mode[k]
Analogous to trig_delay_mode. The THIS and PREV flags do however not apply.

trig_mux_stretch[k]
Analogous to trig_stretch.

trig_lmu_and[j] and **trig_lmu_nand[j]**
One bit (1 << i) in each register for each i -> j connection, together with 1 in bit j of trig_lmu_not gives [and,nand]:

[0,0] - don't care,
[0,1] - require coincidence,
[1,0] - require anti-coincidence,
[1,1] - don't use.

Together with 0 in bit j of trig_lmu_not:

[0,0] - no output,
[0,1] - or of negated input,
[1,0] - or of input,
[1,1] - always on.

trig_lmu_mux_and[j] and **trig_lmu_mux_nand[j]**
Analogous, for the multiplexed inputs, i.e. bits (1 << k)

trig_lmu_aux_and[j] and **trig_lmu_aux_nand[j]**
Analogous, for the auxiliary inputs, i.e. bits (1 << l).

trig_lmu_not
j bits of inversion for the outputs. With bit (1 << j), output j constructs an AND condition (coincidence) instead of an OR condition. A zero bit is must be used to disable an output which has no coincidence requirements at all.

tpat_enable
Bitmask of triggers (j) to enable.

trig_red[j]
Select the downscale factor 2^n.

sum_out_stretch
Length of the master start signal.

sum_out_mask
Bitmask telling which module outputs should see the sum_out (master start). Special multiplexer bypass to be fast!

pending_prompt
Bitmask of pending triggers that instead act as pulsed triggers, i.e. are only taken if requested when the trigger state machine is IDLE.

OUTPUT REGISTERS

These scalers (32-bits) are latched on each event:

before_lmu[i]
Pulses before the LMU.

before_lmu_mux[i]
Pulses before the LMU, multiplexed inputs.

before_lmu_aux[i]
Pulses before the LMU, auxiliary inputs.

before_deadtime[j]
Pulses after the LMU / before deadtime veto.

after_deadtime[j]
Pulses after deadtime veto.

after_reduction[j]
Pulses after reduction.

PULSE REGISTERS

Write access to these registers execute actions:

trig_pending
: Set pending triggers for the bit-pattern written.

trig_clear_pending
: Remove pending triggers for the bit-pattern.

pulse Actions as per (**TRLO_BIG_PULSE_***xxx*):

TRIG_SCALER_RESET - reset the trigger scalers.

TRIG_SCALER_LATCH - latch the trigger scalers.

MASTER_START - fire one master start pulse. (Useful to generate gates for special triggers, e.g. pedestal determination.)

CONTROL SIGNALS

Several control signals are provided to and from the multiplexer.

To the fast-path:

TRIG_LMU_AUX(l)
: Auxiliary LMU input l.

TRIG_LMU_TEST
: LMU test input.

From the fast-path:

TRIG_LMU_OUT_ENABLED_OR
: Or of the enabled LMU output signals.

MASTER_START
: Generated master start signal. (Use sum_out_mask for fast output signals.)

TRIGGER STATE MACHINE

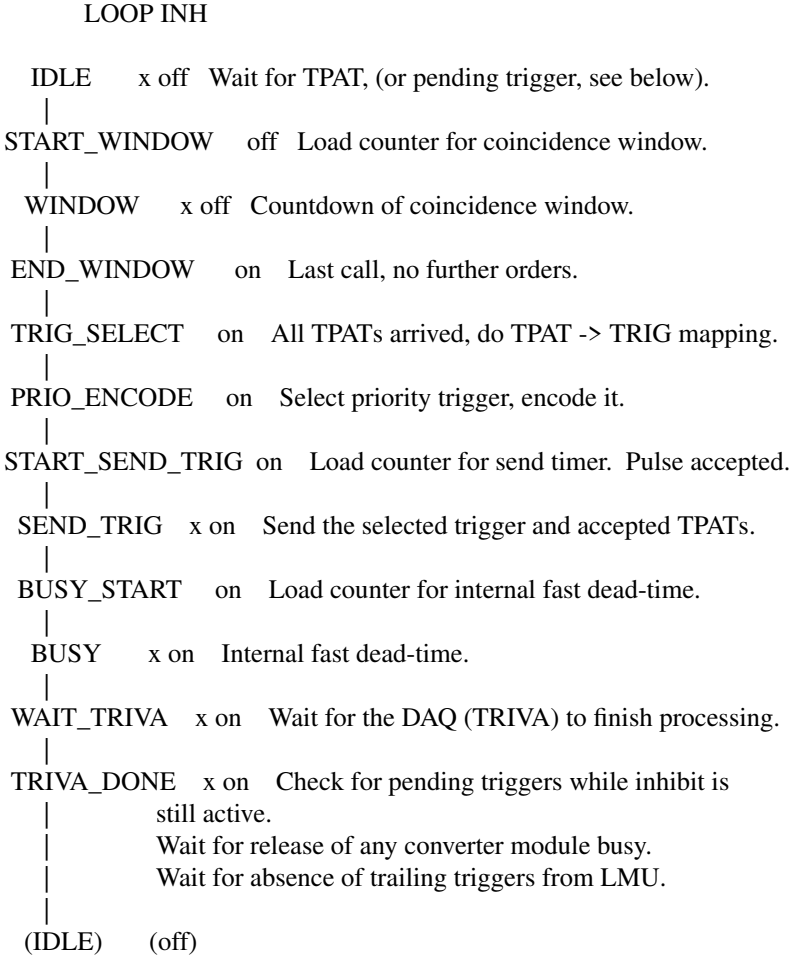
The data acquisition cycle control is in charge of the system dead-time, which is both produced internally (quickly in response to accepted triggers to prevent multi-triggering) and externally (from TRIVA, and busy from converter modules).

Once the DAQ is in running mode, it is normally in the IDLE state. When a trigger has managed to pass through the fast_path and survived downscale, it becomes an accepted trigger. The accepted trigger makes the trigger state machine go into the (programmable) coincidence acceptance window, during which further TPATs are accepted. As the window reaches the end, the internal dead-time (inhibit) is activated to prevent fast_path from letting further TPATs through. When using multiple TPATs, the acceptance window should be long enough to allow for any timing jitter present between different coincidence conditions that may be

related to the same event. Inhibit will then be on until we reach IDLE mode again.

After no further TPATs can arrive (a few cycles are after the acceptance window closure are allowed for, to let the inhibit propagate to the trigger state machine TPAT input), the mapping from received TPATs to TRIVA trigger numbers (1-15) is done. Then the highest trigger is selected (priority) and also encoded, to be sent to the TRIVA (send time is 10 cycles, i.e. 100 ns). Note that the priority encoding in most cases rather should be thought of as defining a clear rule for tie-breaking in the rare cases of multiple triggers being generated within the acceptance window, than that some triggers are more important.

To allow the TRIVA (or equivalent) time to deliver its dead-time, an internal (programmable) busy counter is started. Once this reaches the end, the WAIT_TRIVA state is entered, where it stays until the external dead-time (from the TRIVA) is released. Upon release of the dead-time, it is checked that busy inputs also are clear and that no trailing end of a trigger is present at the LMU output. After this, IDLE state is again active. The start-up mode for the state machine is WAIT_TRIVA.



For multi-event operation, note that it is allowed for a TPAT to map to TRIVA trigger 0, i.e. in this case the TRIVA will not see a trigger at all and only the internal dead-time as well as busy signals wired from the converter modules will hinder the system from reaching IDLE mode again.

Note that the busy from multi-event modules must be wired to the busy input and not to the dead-time input, as the (TRIVA) dead-time will prevent pending triggers from being treated, while the busy input will allow pending triggers to be processed. And the pending trigger is what would be used if a module with an IRQ signals that it is full and therefore needs readout. (Such full modules are expected to issue continuous

busy signals until at least one event is read out.)

TRAILING TRIGGERS

When releasing the inhibit (i.e. allowing triggers to pass from the LMU to the reduction counters of the fast_path), there is a chance that an event just happened some while ago (such that its coincidence signals are still present).

If it was very recently, i.e. only a few ten ns ago, and if the leading-edge detect of fast_path would be before the inhibit veto, we would loose any TPAT bits that correspond to LMU outputs that was then vetoed, but later ones would survive. I.e. we would get a wrong TPAT. Thus, leading-edge before inhibit veto is not good.

Likewise, if the trigger was longer ago, but the leading-edge detect of fast_path would be after the inhibit veto, we would in this case possibly also loose TPAT bits, for those with shorter coincidences. Moreover, this event will likely miss to record any good times, as the master start will be generated unusually late. Ergo, leading-edge after inhibit veto is also not good.

To remedy this, the trigger state machine will not pass from TRIVA_DONE to IDLE mode (i.e. release the inhibit) while any signal is still active from the LMU. Incomplete triggers are thus ignored by extended dead-time. As this affects different kinds of events equally, it does not affect the ratios of collected events. To get complete TPATs in the case where a new trigger comes just the cycle after the inhibit was released, the leading-edge detect is placed after the inhibit veto. This allows these signals (that then are delayed by at most 1-2 cycles) to also be completely recorded.

Note: this has the consequence that if any incoming fast_path detector signal is so noisy that it is constantly on, _and_ if it uses the TRLO_BIG_TRIG_DELAY_MODE_WHEN_PRESENT mode, it can completely block the acquisition by never allowing the inhibit to be released. (The display and VME registers can be used to detect such situations.)

PENDING TRIGGERS

The exception to the normal control cycle is the occurrence of pending triggers, either software or hardware generated. A pending trigger is a request to generate a particular trigger, which will not go away until that particular trigger has been generated and accepted by the priority encoder. There are two ways of reaching these triggering states, depending on if the system was IDLE when the request was made.

If non-idle, i.e. already processing a trigger, then a check for pending triggers is made after the TRIVA has released its dead-time, but before we have released the inhibit to the fast_path, i.e. there is no chance for any TPATs to be accepted.

```
 TRIVA_DONE      on    Check for pending triggers while inhibit is
    |                  still on.
    |
PULSE_SELECT     on    Triggers requested are those that are pending.
    |
(PRIO_ENCODE)   (on)   Select priority trigger, as usual.
```

If the system is in IDLE mode as the trigger arrives, there exist the chance of a real (detector) trigger arriving and sneaking through the fast-path while accepting the pending trigger. The issue is that if it is accepted by the fast_path, a master start will have been generated, which may not be compatible with the wishes of the pending trigger. This is handled by realising that the pending trigger can wait until the accepted trigger has been dealt with the normal way. This is done with states resembling those that follow the closure of the normal coincidence window. If a TPAT is seen, control is transferred to the normal closing

procedure, and the ordinary trigger is handled.

A pulsed trigger is another way of making a trigger, but it will only be accepted if the system is IDLE while the pulse happens. This is also subject to the safety measures preventing spurious master starts. Note: pulsed triggers by themselves are currently only available as pending triggers marked as prompt.

```
   IDLE        off   Pending (or pulsed) trigger is check for.
    |
PENDING_PULSE_TRIG     Wait in case TPAT sneaked through.
    |         on    (if it sneaked, go to TRIG_SELECT)
    |
 PULSE_SELECT    on    Triggers requested are those that are pending
    |              or pulsed.
    |
(PRIO_ENCODE)   (on)   Select priority trigger, as usual.
```

SUDDEN DEAD-TIME

If the TRIVA suddenly starts DT, the best we can do is to go to WAIT_TRIVA state and wait for the DT to go off. Even though we could check for this condition also in START_WINDOW and WINDOW, it will not prevent spurious triggers from leaking through in those cases... So only the cases that will anyhow not (soon) end up in WAIT_FOR_TRIVA must do this check, i.e. IDLE and TRIVA_DONE.

It is ensured that all master start signals that are issued are accompanied with an taken trigger and thus also an accept pulse. For that reason, if an master start is issued at the same time as an suddenly appearing deadtime or busy signal, the system will linger for one cycle in SUDDEN_DT or SUDDEN_BUSY and go to START_WINDOW instead of WAIT_TRIVA.

```
   IDLE        off   Sudden deadtime and busy is checked for.
    |
 SUDDEN_DT or         Wait one cycle in case TPAT sneaked through.
 SUDDEN_BUSY     on    (if it sneaked, go to START_WINDOW)
    |
 WAIT_TRIVA      on    Wait for the DAQ (TRIVA) to finish processing.
```

Note that in both cases, if the DAQ or acquisition modules actually are busy, the thus generated trigger may not be completely processed. Such errors should be caught by the readout programs, and errors mean that either the deadtime or busy cabling is leaky.

This also applies to the 'stop acquisition' software generated trigger 15 event. Using also the GO bit from the TRIVA would not help. Even if the acquisition software as the first action of 'stop acq' removes the GO bit and then continuously holds either DT or !GO, during the setting of !GO, we may generate a spurious master start... Therefore, in order to have a completely leak-free system, it would be necessary to send the stop acq trigger 15 via us... No other way.

This just shows that one and only point must act as the hour-glass waist in terms of fan-in and fan-out of the system dead-time, in analogy to what also applies to the master start for common timing, being distributed to all systems. As far as the master start goes, the output of the TRLO_BIG is not that point! (Since e.g. the TCAL module start is fanned-in at a later point.)

CONTROL REGISTERS

tpat_trig[j]

For each tpat (j) determine which trigger (i) (if any) it provokes. Use 0 for multi-event mode.

max_multi_trig

Limit the number of events that do not produce a trigger (tpat_trig[j] = 0). 0 = disable. Note that any trigger resets the internal counter, i.e. readout must be performed on all triggers.

multi_trigger

Fire this trigger upon reaching max_multi_trig.

accept_window_len

Length of the coincidence acceptance window, 10 ns units.

fast_busy_len

Length of the internally generated dead-time, 10 ns units.

trig_control

The global control register has two bits telling if an accepted trigger should set the internal dead-time and/or busy flip-flops. Intended for testing purposes (**TRLO_BIG_TRIG_CONTROL**_*xxx*):

ACCEPT_SETS_INTERNAL_DT - Set internal DT on trigger.
ACCEPT_SETS_INTERNAL_BUSY - Set internal busy on trigger.

The trigger scalers can be reset after latching each accepted event:

ACCEPT_RESETS_TRIG_SCALER - Reset each event.

OUTPUT REGISTERS

trig_time[2]

Latched timing counter of the current accepted event, lo and hi 32-bit word in [0] and [1].

trig_tpat_cnt

TPAT (i.e. LMU out, reduction accepted pattern) of current accepted event. The encoded trigger number is stored in bits 24-27 and a 4-bit event counter in bits 28-31. Bits 22-23 hold a bitmask of which toggling master start output fired.

trig_count

The 32-bit event counter.

trig_checksum

Checksum of the previous two words, useful to verify correct VME transfer. Use only when in dead-time, as the contents otherwise may change while the words are read. To also detect double-errors on individual data lines, it is defined as:

```
trig_checksum =
  ((trig_tpat_cnt  >> 1) | (trig_tpat_cnt  << 31)) ^
  ((trig_count     >> 2) | (trig_count     << 30));
```

trig_status

Bit-mask giving the status of the trigger state machine, and the reasons why it is where it is (**TRLO_BIG_TRIG_STATUS**_*xxx*):

DT_IN - Dead-time input sees signal.
BUSY_IN - Busy input sees signal.
INTERNAL_DT - Internal dead-time flip-flop on.
INTERNAL_BUSY - Internal busy flip-flop on.
DT - Or of DT input and internal ff.
BUSY - Or of busy input and internal ff. (The two previous to be removed.)
AFTER_LMU_ENABLED_OR - Any signal after LMU is present now.
LMU_STUCK_OR - Any signal after LMU has been present for more than 100 us (prevents state-machine from going idle).
LMU_ENABLED_STUCK_OR - Any enabled signal after LMU has been present for more than 100 us.
INHIBIT - State-machine current generating deadtime, i.e. system deadtime.
STATE_MASK/SHIFT - 5-bit value representing the current state.
REASON_MASK/SHIFT - 4-bit value telling why the current trigger was taken.

The values describing the STATE above are (**TRLO_BIG_TRIG_STATUS_STATE**_*xxx*):

IDLE - Idle.
START_WINDOW - TPAT coincidence window.
WINDOW - TPAT coincidence window.
END_WINDOW - Coincidence window closed.
TRIG_SELECT - Do TPAT -> TRIG mapping.
PRIO_ENCODE - Select highest trigger.
START_SEND_TRIG - Send trigger.
SEND_TRIG - Send trigger.
BUSY_START - Internal fast dead-time.
BUSY - Internal fast dead-time.
WAIT_TRIVA - Wait for dead-time release.
TRIVA_DONE - Wait for busy release.
PENDING_PULSE_TRIG - Take pending trigger.
PULSE_SELECT - Select pending trigger.
SUDDEN_DT - (Unexpected) dead-time during idle.
SUDDEN_BUSY - (Unexpected) busy during idle.

The values describing the REASON above (to have inhibit on) are (**TRLO_BIG_TRIG_STATUS_REASON**_*xxx*):

IDLE - Idle (no inhibit).
TRIGGER - Normal trigger taken.
PENDING_TRIG - Pending trigger taken.
PULSE_TRIG - Pulse trigger taken (unused).
DT_ON_IDLE - (Unexpected) dead-time during idle.
BUSY_ON_IDLE - (Unexpected) busy during idle.
DT_ON_BUSY - (Unexpected) dead-time (again) while waiting for busy release.
PEND_IN_BUSY - Pending trigger taken while waiting for busy release.
TRIG_ON_PEND - Normal trigger overruled pending trigger that was about to be taken.
TRIG_ON_SUD_DT - Trigger overruled (unexpected) dead-time state.
TRIG_ON_SUD_BUSY - Trigger overruled (unexpected) busy state.

pending
Bit-mask of triggers still pending.

lmu_stuck_in
Technically from fast_path. Bit-mask of which LMU inputs that are stuck active for more than 100 us.

lmu_stuck_out
As above, but for LMU outputs, also see LMU_STUCK_OR above.

lmu_enabled_stuck_out
As above, but only including enabled LMU outputs, also see LMU_ENABLED_STUCK_OR above.

CONTROL SIGNALS

Several signals are provided to and from the multiplexer.

To the state machine:

TRIG_PENDING(i)
With the leading edge of a pulse, the pending bit for trigger (i) is set.

DEADTIME_IN(i)
Input for dead-time. For convenience, several (2) sources are or'ed together.

BUSY_IN(i)
Input for busy-in.

From the state machine:

ACCEPT_TRIG(i)
Signal with the accepted trigger, only one bit at a time. Signal is 10 cycles long, i.e. 100 ns. (0) indicates an multi-event trigger 0.

ENCODED_TRIG(i)
Signal with the encoded accepted trigger, suitable to be sent to the TRIVA. The signals are 10 cycles long.

DEADTIME
The system deadtime.

ACCEPT_PULSE
1-cycle pulse after an accepted trigger.

PULSE REGISTERS

pulse Actions as per (**TRLO_BIG_PULSE_***xxx*):

SET_INT_DT - set internal deadtime flip-flop

CLEAR_INT_DT - clear internal deadtime flip-flop

SET_INT_BUSY - set internal busy flip-flop

CLEAR_INT_BUSY - clear internal busy flip-flop

MULTI-TRIGGER BUFFER (TPAT AND TIME)

The time (63 bits) and TPAT/trigger/count (i.e. contents of trig_tpat_cnt, see below) are for every trigger stored as three 32-bit words into a multi-entry buffer. The first word contains the low 32 bits of the time. The second holds the next 31 bits together with an overflow marker in the highest bit. The third word contains the TPAT/toggle/trigger/count. The buffer has 512 entries and thus can store information of up to 170 triggers. If it becomes full, and trigger information thereby cannot be stored and is lost, the following stored trigger will have the high bit of the second word (time hi) set to one. A control word allows to set an 'almost-full' level, at which an output at the multiplexer will go active. This can be connected to a pending trigger to enforce readout, even if using the multi-event (trigger = 0) mode.

Reading from any address within the output array will provide the next value from the FIFO. A status word with the number of 32-bit words (not triggers) left must be consulted before reading, as there is no unique no-valid-data marker (0x5a5aa5a5 will however be delivered). The status word also has a 16-bit XOR checksum (XOR of low and high 16-bit word parts) of the data currently in the buffer. After reading the number of data words stated, the checksum read together with the word count should match the data. If the checksum is non-zero when no more data words are available, then the internal memory array has suffered a bit-flip.

multi_trig_buf_control

Almost-full level of the multi-trigger buffer (**TRLO_BIG_MULTI_TRIG_BUF_CONTROL**_*xxx*):

ALM_FULL_LEVEL_MASK/SHIFT - Almost full level. Use ALM_FULL_LEVEL(n) to set value, with n='max' for safe value, or 'half'.

multi_trig_buf_status

Status of the multi-trigger buffer, number of 32-bit words available and XOR checksum (**TRLO_BIG_MULTI_TRIG_BUF_STATUS**_*xxx*):

DATA_AVAIL_MASK/SHIFT - Available words.
CHECKSUM_MASK/SHIFT - Checksum.

multi_trigbuf[]

Next data word from multi-trig buffer.

MULTI_TRIG_BUF_ALM_FULL

Multi-trigger data output buffer is almost full.

pulse Actions as per (**TRLO_BIG_PULSE**_*xxx*):

MULTI_TRIG_BUF_CLEAR - clear the multi-event buffer

MULTI-EVENT SCALER BUFFER

The generic and trigger-related scalers can be copied into a multi-event buffer on each trigger. This is particularly useful in multi-event mode, as otherwise the scalers would only be read for the last event. A downscale can be applied to the multi-events (trigger-0), reducing the amount of data recorded. Scalers are always copied on non-0 triggers.

The scaler values are copied during idle cycles of the normal register access interface. Copying takes about 1 us. A status bit reports if sequencing is currently active.

The first word of each event contains the trigger number in the low 16 bits. The highest bit marks if an scaler latch was ignored due to buffer bull. This is then followed by the selected data, in the same order as the scaler registers. There are no markers identifying the individual values.

A signal is provided such that other multi-event scaler modules can be latched at the same times (events) as the multi-event scaler records data.

multi_scaler_use_gen
Bitmask selecting generic scalers for inclusion.

multi_scaler_use_before_lmu
Bitmask selecting before-LMU scalers.

multi_scaler_use_before_mux_lmu
Bitmask selecting before-LMU scalers, multiplexed inputs.

multi_scaler_use_before_aux_lmu
Bitmask selecting before-LMU scalers, auxiliary inputs.

multi_scaler_use_tpat
Bitmask selecting after-LMU scalers to include (before and after deadtime, and after reduction).

multi_scaler_downscale
Downscale factor for trigger-0 events.

multi_scaler_control
Almost-full level of the multi-event scaler buffer (**TRLO_BIG_MULTI_SCALER_CONTROL_***xxx*):

ALM_FULL_LEVEL_MASK/SHIFT - Almost full level. Use ALM_FULL_LEVEL(n) to set value, with n='max' for safe value, or 'half'.

multi_scaler_status
Status of the multi-event scaler buffer, number of 32-bit words available and XOR checksum (**TRLO_BIG_MULTI_SCALER_STATUS_***xxx*):

DATA_AVAIL_MASK/SHIFT - Available words.
CHECKSUM_MASK/SHIFT - Checksum.
SEQUENCING - Sequencer is running.

multi_scaler[]
Next data word from multi-scaler buffer.

MULTI_SCALER_ALM_FULL
Multi-event scaler data output buffer is almost full.

MULTI_SCALER
Signal fires when the multi-scaler starts recording. Use to latch slave scaler modules.

pulse Actions as per (**TRLO_BIG_PULSE_***xxx*):

MULTI_SCALER_CLEAR - clear the multi-event buffer.

MASTER START PING-PONG MODE

When operating in multi-event mode, most of the conversion time of events can be hidden, if every signal is connected to two modules. Each pair of modules is operated in a ping-pong fashion, such that only one of them receives a gate to initiate conversion for each event. Thus, the other module in the pair is ready to accept the next event before the first has finished conversion. Only when both modules are busy converting, the entire system is busy.

When the setup has some modules with long conversion times and some with short conversion times, an improvement can already be attained if the modules with long conversion times are operated in ping-pong fashion.

The TRLO_BIG makes this mode of operation possible by monitoring the busy signals of the two sets of modules (each set contains one module from each pair). It generates ping-pong master start signals for the sets in an alternating fashion. However, subsequent master starts can be sent to one set if the other set is continuously busy. This could happen if some module in the latter set e.g. reports busy due to a full multi-event buffer.

Since data from each channel will be recorded alternatingly in two separate digitisers, it is for calibration purposes very useful to have some of the events recorded by both digitisers. TRLO_BIG supports this by a selectable downscale, where every n'th event fires both sets of modules. This naturally causes a system-wide busy for the conversion time duration of that event. In order to somewhat relax the impact, the dual-set firing is postponed for up to n events, while one of the branches is busy when the original master starts arrive.

If the setup is operated both with ping-pong pairs of modules, and some modules (having shorter conversion times) that record all events, then the busy signals of the latter class of modules shall be connected such that they are considered for both sets of ping-pong modules.

Information of which (or both) of the toggle sets that fired for each event is available in the multi-trigger buffer, as well as the trig_tpat_cnt output register.

busy_toggle_control
A combination of two settings (**TRLO_BIG_BUSY_TOGGLE_CONTROL_***xxx*):

Enable toggle functionality:

ENABLED - In use.

Recording of events in both sets of modules:

DOWNSCALE_BOTH_MASK/SHIFT - Downscale factor. Use DOWNSCALE_BOTH(n) to set value, n=0 disables dual recording

BUSY_IN(i)

When toggle-mode is enabled, the two busy inputs relates to the two sets of modules. The trigger state machine sees a busy when both sets report busy. (Note that the fast deadtime of the trigger state machine prevent triggers from being taken immediately after each other.)

When disabled, the two busy inputs are or'ed together.

MASTER_TOGGLE(i)

Master start signal for the each set of modules. At least one of these signals will be generated for each master start.

Modules which are not operated in the ping-pong set should use the normal master start signal.

When disabled, both outputs replicate the master start signal.

Scalers for monitoring and cross-check purposes:

master_start_toggle[i]

Counts the number of master starts issued to each set of modules.

These scalers are latched together with the trigger scalers.

SYNC CHECK

To check synchronisation between systems, a signal of variable length or delay can be used. This is inspired by the Time Generator functionality of the GSI Exploder module [x]. The length or delay is given (chosen) by the sending system using its local event counter and random bits. Each receiving system measures the length or delay. When the data has been combined (e.g. using timestamps) it can then be verified that the measured values in all systems correlate (perfectly) for data that is associated with the same physical event.

The variation is controlled by six bits. The two lower bits are random, while the upper four bits are the lower bits of the trigger counter. With a bitmask it can be chosen which of the bits actually should be used. (A bitmask of 0 disables this variation functionality.) The lowest bit corresponds to a pulse length or delay variation of 40 ns, the next bit 80 ns and so on. Currently the length of the master start signal is varied. (A delayed signal can be obtained by using the generic LMU.)

A receiving system can measure the variation using the built-in measurement function, or a TDC. Since the variation steps are large, only limited resolution is required. An ADC or QDC can also be used by applying a suitable RC filter, converting the pulse length into a pulse height.

The sync check can also be encoded in the output of the first two generic LMU gate generators.

CONTROL REGISTERS

sum_out_scadd_bits
Limit which bits are used to vary the master start pulse length.

sync_check_start_mux
Select module input for the measurement start of the sync check value, i.e. one of **TRLO_BIG_MUX_SRC**_*xxx* that marks a real module input.

sync_check_stop_mux
Select module input for the measurement stop of the sync check value, i.e. one of **TRLO_BIG_MUX_SRC**_*xxx* that marks a real module input.

sync_check_mode
It is individually selectable if the measurement starts and stops with leading or trailing edge.

For the start (**TRLO_BIG_SYNC_CHECK_MODE_START**_*xxx*);

LEADING_EDGE - measure from leading edge (typical use),
TRAILING_EDGE - measure from trailing edge.

For the stop (**TRLO_BIG_SYNC_CHECK_MODE_STOP**_*xxx*);

LEADING_EDGE - measure to leading edge (typical for delayed stop pulse),

TRAILING_EDGE - measure to trailing edge (typical for pulse length measurement).

trig_latch_delay
Delay the latching of the measured value into the trig_sync_check output register. The delay starts when the accept pulse of a trigger has been generated, i.e. at the end of the trigger window.

This also delays the filling of the multi-event trigger buffer.

OUTPUT REGISTERS

trig_sync_check
Contains the measured pulse length or delay in the lower 10 bits.

Bits 16-21 contain the value used to generate the most recent variable length master start.

SOFT-SCOPE (TRACER)

The tracer can provide information about the time-wise relationship between signals, much like an oscilloscope. It runs autonomously - recording is self-triggered by user-defined signal coincidences

The tracer first stores the input bit-pattern on every clock cycle into a circular buffer provided by a memory block (128 entries x max 24 values). This runs continuously until and beyond a trace-trigger condition is detected. On a trigger, the compactification starts from the beginning of the buffer, i.e. ˜128 entries back and continues for 256 cycles. The trigger condition thus ends up at the middle (minus 2 for technical reasons, i.e at 126).

The 24 values for the trigger tracer are the trigger signals after the delay and stretcher, before the LMU starting at bit 0. The aux trigger signals are available starting at bit 20.

The information to be read out is compressed by only copying the patterns that change, together with their time information into a new memory buffer, with the time in the high 8 bits, and the bit-pattern in the lower bits. This way, the number of VME transfers are reduced (compared to reading the full non-changing history). The compressed trace always begins with a 31-bit time-stamp corresponding to the time of the trigger condition, marked with a 1 in the highest bit. Then follows the state at time 0, and then for all times when the pattern changed. It ends with a checksum (16 bit xor of the two halves of data words, right-rotate-1-xor for each item) of the data, marked with a full 16 bit zeroes in the high bits. This distinguishes against the data as any data following the first pattern (with time 0) has a non-zero time-stamp.

The maximum length of a stored trace is 258 32-bit words. The compacted buffer at most can handle 1024 entries. By only starting to store data if it has < 512 items, the check can be done at one place before accepting a new trigger without any risk of overflow during compression. If full condition is reached, the tracer goes into idle state, and has to be restarted via VME (after some readout or clearing).

Once the compactification copying is done, the tracer can immediately trigger again, as the ring-buffer was filled during compactification. Due to the compactification into a second RAM block, many traces can be collected autonomously.

```
    IDLE        Waiting for start command from VME.
     |
    START       Starting.  If buffer space insufficient, go to idle.
     |
  INIT_FILL     Make sure ring buffer has recent values, by looping
     |          for 128 cycles.
     |
   ACTIVE       Wait for coincidence condition.
     |
 COINCIDENCE    Coincidence found.  Store current time-stamp.
     |
COMPACTING_FIRST  Store first bit-pattern, i.e. value stored into
     |          ring buffer 128 cycles ago.
     |
  COMPACTING    For 255 cycles, check if ring buffer value stored
     |          128 cycles ago differs from last bit-pattern written
     |          to output buffer.  If so, store.
     |
   COMPACTED    Store checksum.  Next state is start.
```

The trigger condition is controlled by setting a number of multiplexers, which will monitor their selected inputs. As each input sees a signal, a local shift-register flag is set, which will persist until cleared. Clearing happens every n cycles, and the shift register has four slots, allowing a maximum coincidence window of 3n to 4n-1, depending on when the flag was set. n is a user parameter, at most settable to 255. Values larger than 31 may cause parts of the coincidence condition to be earlier than the first sample stored.

By setting several multiplexers to the same source, the number of required coincidences are reduced (as they become self-coincidences). Each multiplexer can be set as an anti-coincidence requirement by setting its 6th bit. The coincidence shift registers cannot be explicitly cleared, but by setting the clear clock to 1 (or 0), they will be flushed within 4 cycles, i.e. for all practical purposes immediately. (This is much less than the minimum VME access time.)

The only control register holds the 5-bit multiplexer values, the anti-coincidence bits as well as the coincidence timeout power. Writing to it immediately changes the coincidence requirements, but does otherwise not affect any running acquisition. The tracer is started by a VME pulse, and can also be stopped/cleared by another.

For every read-out anywhere in the read-out array, the read-out pointer is advanced by one. One register holds the number of values still available in the buffer, and must be consulted before readout, as there exist no unique buffer-is-empty special value. It also has two bits telling if data collection is active, and if a trace is being compacted, respectively. If the latter is active, the number of available data words will increase, as at least the checksum will be appended. Spurious reads on an empty buffer will not move the internal read pointer.

CONTROL REGISTERS

tracer_control

Select trigger signals and condition. Set by a combination (or) or:

TRLO_BIG_TRACER_COINC_MUX(i,no) selects signal no for coincidence mux i

TRLO_BIG_TRACER_ANTI_COINC(i) sets anti-coincidence requirement for mux i

TRLO_BIG_TRACER_COINC_COUNTER(t) set the coincidence shift-register shift interval to t cycles

OUTPUT REGISTERS

tracer_status

The number of 32-bit data words available in the output buffer, if the tracer is active, and if it is compacting data (**TRLO_BIG_TRACER_STATUS_***xxx*):

DATA_AVAIL_MASK/SHIFT - Available words.
ACTIVE - Tracer is active (can trigger).
COMPACTING - Tracer is compacting a trigger now.

tracer[]

Next data word.

PULSE REGISTERS

pulse Actions as per (**TRLO_BIG_PULSE_***xxx*):

TRACER_START - start tracing, i.e. start looking for a coincidence

TRACER_CLEAR - stop tracing and clear the readout buffer

MULTIPLEXER

The TRLO_BIG is also equipped with several other small function blocks. This is to besides direct triggering also allow it to perform some other experiment-specific decisions and recordings based on (other / related) logic signals. To allow general routing of the signals from the inputs to these functions, and then to the trigger cycle control and vice versa, each consumer can choose whichever signal producer to use as a source for its actions.

This means that each consumer is fed by a multiplexer sourcing all producers. This is implemented by aliasing all producers and consumers as two arrays, respectively. The setup register for each destination tells which source to use (with xxx from the tables below):

mux[**TRLO_BIG_MUX_DEST_***xxx*(j)] = **TRLO_BIG_MUX_SRC_***xxx*(i);

Destinations (j): **TRLO_BIG_MUX_DEST_***xxx*(j)

module outputs (VULOM):
ECL_OUT
ECL_IO_OUT
LEMO_OUT

module outputs (TRIDI):
CTRL_OUT
TRIG_BUS_OUT
NIM_OUT

front-panel:
FRONT_LED

logic functions (inputs):
LMU_IN
GATE_DELAY
EDGE_GATE_START
EDGE_GATE_STOP
DOWNSCALE

recording:
SCALER
SCALER_LATCH
SCALER_RESET
TIMER_LATCH
MULTI_LATCH_DISCARD_OLD
PTN_LATCH

serial timestamp:
SERIAL_TSTAMP_IN
SERIAL_TSTAMP_LATCH
SERIAL_SIGNALS_IN

fast-path & trigger control:
TRIG_LMU_AUX
TRIG_LMU_TEST
TRIG_PENDING
DEADTIME_IN
BUSY_IN

random tcal:
RANDOM_TCAL_START
RANDOM_TCAL_STOP

Sources (i): **TRLO_BIG_MUX_SRC_***xxx*(

module inputs (VULOM):
ECL_IN
ECL_IO_IN
LEMO_IN

module inputs (TRIDI):
CTRL_IN
TRIG_BUS_IN
NIM_IN
ECL_IN

generators:
WIRED_ZERO
WIRED_ONE
PRNG_LFSR
PRNG_POISSON
PULSER

logic functions (outputs):
LMU_OUT
GATE_DELAY
EDGE_GATE
DOWNSCALE
ALL_OR
COINCIDENCE
INPUT_COINC

rec:
MULTI_LATCH_ALM_FULL

serial timestamp:
SERIAL_TSTAMP_OUT
SERIAL_TSTAMP_ALM_FULL
SERIAL_TSTAMP_DESYNC
SERIAL_SIGNALS_OUT

heimtime:
HEIMTIME_OUT

trigger control:
ACCEPT_TRIG
ENCODED_TRIG
MASTER_START
DEADTIME
ACCEPT_PULSE
TRIG_LMU_OUT_ENABLED_OR
MULTI_TRIG_BUF_ALM_FULL
MULTI_SCALER_ALM_FULL

trimi:
TRIMI_TDT

Additionally, each module output can be directly connected to the signal of any module input, without clocking (latching), in order to transport timing signals. This would be mostly of interest to the TRIDI together with its signal bus. Together with this, the direct input can alternatively be ANDed with the selected clocked output (that can come from any function generator) to allow for logical conditions deciding if the signal should be sent at all. By making sure that the direct non-latched signal comes after the logic decision, the timing would be defined by the direct signal.

Implementation note: the multiplexing unfortunately uses two clock cycles. This is due to the large fan-out required for each source in combination with the rather long multiplexing chains.

To partially overcome the connection latency, the outputs of some functional units (and module inputs) and be directly connected to the input of some (other) functional units (and module outputs). These direct

connections go between signal numbers with the same index only. The inputs are or'ed together with any signal chosen via the multiplexer.

CONTROL REGISTERS

mux[j] Tell which source (i) should be used by each destination (j), i.e. one of **TRLO_BIG_MUX_SRC_***xxx*.

j is one of **TRLO_BIG_MUX_DEST_***xxx*.

nonlatched_mux[k]
Tell which non-clocked module input (l) should be used by module output (k), i.e. one of **TRLO_BIG_MUX_SRC_***xxx* that marks a real input (not a function generator).

k is one of **TRLO_BIG_MUX_DEST_***xxx* that marks a real output.

nonlatched_or[m]
Bitmask telling which unclocked module inputs (1 << l) should be or'ed together to form direct OR m.

nonlatched_mode[k]
Two bits for each output (**TRLO_BIG_NONLATCHED_MODE_***xxx*):

LOGIC - use the clocked mux[k] source,
NONLATCHED - use the non-clocked direct_mux[k] source,
LOGIC_OR_NONLATCHED - OR of any of the two sources,
LOGIC_AND_NONLATCHED - AND of the two sources.

The direct input can be either of (**TRLO_BIG_NONLATCHED_IN_***xxx*):

MUX - one multiplexed input
OR(m) - an OR of several inputs

direct_mask[j][i]
Bitmask telling which signals (indices) of the destination functional unit (j) shall receive direct signals from source functional unit (i).

Destination (j) is one of (**TRLO_BIG_DIRECT_DEST_***xxx*):

ECL_OUT - module output,
ECL_IO_OUT - module output,
LEMO_OUT - module output,
LMU_IN - logic matrix,
DOWNSCALE - downscale unit.

Source (i) is one of (**TRLO_BIG_DIRECT_SRC_***xxx*):

ECL_IN - module output,
ECL_IO_IN - module output,
LEMO_IN - module output,
LMU_OUT - logic matrix,
DOWNSCALE - downscale unit,

INPUT_COINC - module input coincidence unit.

Note that the master start can be connected faster (i.e. without the multiplexing delay) with the sum_out_mask control register (see fast_path) to (m)any module output(s).

GENERAL LOGIC FUNCTIONS

The logic functions uses one or more signals as given by the multiplexer to generate new signals that again can be accessed by the multiplexer. They are described in the following, with their respective control registers (the index i just denotes that there are several of each kind):

PULSER

Generate a one-clock pulse every period clock cycles. Please use a gate-and-delay generator to make it longer.

To synchronise a pulser to some external periodic event, first set a start time and then set which pulser(s) to restart at that time.

pend_restart_wait
Bitmask of pulsers waiting for restart.

period[i]
Period in 10 ns steps. (**TRLO_BIG_PERIOD**_*xxx*):

VALADD - The minimum period (encoded by a 0 value).

restart_at
Restart selected pulsers at given time (c.f. timing_tick[0]).

restart_wait
Set pulsers (using the bit-pattern written) to restart at the given time.

clear_restart_wait
Clear pending restart of selected pulsers.

PULSER(i)
Output pulses i.

PRNG

Pseudo-random sequence generator. Uses a LFSR (linear feedback shift register). The two units are independent with periods 2^63-1 and 2^60-1.

prng_period[i]
Period of the PRNG update.

PRNG_LFSR(i)
Output signal pattern i.

PRNG POISSON PROCESS

Poisson process driven by a 64-bit xorshift PRNG. A pulse is emitted when the low 32-bits of the random value is smaller than a configured value.

prng_poisson[i]
Chance to emit a pulse.

PRNG_POISSON(i)
Output signal.

DOWNSCALE

Generate an output signal every n'th (leading edge) pulse.

downscale[i]
Downscale factor n.

DOWNSCALE(i)
Input signal i to downscale.

DOWNSCALE(i)
Output downscaled signal i.

DELAY & STRETCH

Independently delay and stretch a signal. The delay-line is implemented as a shift register, i.e. will not loose pulses.

delay[i]
Delay.

stretch[i]
Output signal length.

restart_mode[i]
Restart mode of the stretcher (**TRLO_BIG_RESTART_MODE_***xxx*):

LEADING_EDGE - leading edge of pulse
TRAILING_EDGE - trailing edge of pulse
LEAD_IF_INACT - leading edge of pulse, if not already active
WHEN_PRESENT - as long as pulse is active

GATE_DELAY(i)
Input signal i to delay and stretch.

GATE_DELAY(i)
Output pulses i.

LMU

A logic matrix behaving the same as the one for the fast_path (see that for details).

lmu_and[j] and **lmu_nand[j]**
One bit per register for each i -> j connection.

lmu_not
j bits of inversion for the outputs.

LMU_IN(i)
Input signal i.

LMU_OUT(j)
Output signal j.

Each output can optionally be passed through a delay- & stretch-unit, like the pure unit described above. This incurs an additional 2 cycle delay due to the delay unit when enabled.

lmu_delay[j]
Delay.

lmu_stretch[j]
Output signal length.

lmu_restart_mode[j]
Restart mode of the stretcher (**TRLO_BIG_LMU_RESTART_MODE_***xxx*):

LEADING_EDGE - leading edge of pulse
TRAILING_EDGE - trailing edge of pulse
LEAD_IF_INACT - leading edge of pulse, if not already active
WHEN_PRESENT - as long as pulse is active

Enable the delay- & stretch (**TRLO_BIG_LMU_RESTART_GATE_***xxx*):

DISABLE - Not in use, pure LMU output.
ENABLE - In use.

lmu_scadd_step[j]
Add the (trigger) sync check value to the stretched signal length. A non-zero step value (1-3) gives a basic step of 40, 80 or 160 ns.

Only applies to the first two LMU outputs. The delay- & stretch must be enabled.

EDGE-TO-GATE

Convert start- and stop pulses to a long gate. Use to e.g. implement a spill mimic.

As output, one can inspect the state of the flip-flops:

edge_gate
Bit-mask of the edge-gate generator flip-flops.

EDGE_GATE_START(i)
Signal pulse to start generator i.

EDGE_GATE_STOP(i)
Signal pulse to stop generator i.

EDGE_GATE(i)
Output on/off signal i.

The flip-flops can also be pulsed via VME:

pulse As per (**TRLO_BIG_PULSE_***xxx*):

EDGE_GATE_START(i) - start gate for generator i.

EDGE_GATE_START_ALL - convenience bitmask to start all gate generators.

EDGE_GATE_STOP(i) - end gate for generator i.

EDGE_GATE_STOP_ALL - convenience bitmask to stop all gate generators.

FAN-IN (ALL-OR)

Make an OR of selected mux source signals. Useful to e.g. combine many module inputs containing busy-signals from converter modules.

all_or_mask[i][j]
Mask telling which mux sources should be used for OR signal i. j denotes that several 32-bit registers are needed to cover the full mux source array. It is suggested to use the **TRLO_BIG_ALL_OR_***xxx* helper macros.

ALL_OR(i)
Output OR signal i.

COINCIDENCE

Makes an output signal when a sum of selected LMU inputs is equal-or-more than a selected level. The LMU inputs are re-used for inputs as this otherwise would need a lot of multiplexers itself, and is not likely to be used in many cases.

coinc_mask[i]
LMU inputs to consider.

coinc_level[i]
Required coincidence level.

COINCIDENCE(i)
Output coincidence signal i.

A second set of coincidence units use bitmasks of the module inputs.

input_coinc_mask[i]
Bit-mask of **TRLO_BIG_MUX_SRC**_*xxx* signals to consider.

input_coinc_level[i]
Required coincidence level.

INPUT_COINC(i)
Output coincidence signal i.

Each of these outputs can optionally be passed through a delay- & stretch-unit. This incurs an additional 2 cycle delay due to the delay unit when enabled.

input_coinc_delay[i]
Delay.

input_coinc_stretch[i]
Output signal length.

input_coinc_restart_mode[i]
Restart mode of the stretcher (**TRLO_BIG_INPUT_COINC_RESTART_MODE**_*xxx*):

LEADING_EDGE - leading edge of pulse
TRAILING_EDGE - trailing edge of pulse
LEAD_IF_INACT - leading edge of pulse, if not already active
WHEN_PRESENT - as long as pulse is active

Enable the delay- & stretch (**TRLO_BIG_INPUT_COINC_RESTART_GATE**_*xxx*):

DISABLE - Not in use, pure coincidence output.
ENABLE - In use.

MULTIPLEXER PULSING

The multiplexer sources and destinations can be pulsed via VME. First set up the respective bit-pattern arrays, then issue a pulse. It is suggested to use the **TRLO_BIG_ALL_OR**_*xxx* helper macros.

pulse_mux_src_mask[i]
Bit-mask of **TRLO_BIG_MUX_SRC**_*xxx* signals to pulse.

pulse_mux_dest_mask[i]
Bit-mask of **TRLO_BIG_MUX_DEST**_*xxx* signals to pulse.

pulse Fire pulses as per (**TRLO_BIG_PULSE**_*xxx*):

MUX_SRCS - Generate pulses before the multiplexer.

MUX_DESTS - Generate pulses after the multiplexer.

MONITORING

SCALERS

Count events on the input signal. The scalers can be latched in blocks (common for several scalers) on different signals, or by software.

scaler_mode[i]

An combination (or) of two settings:

What kind of events are counted (**TRLO_BIG_SCALER_MODE_***xxx*):

LEADING_EDGE - number of pulses, leading edge
TRAILING_EDGE - number of pulses, trailing edge
DURATION_CLK - total length of pulses, every clock
DURATION_TICK - total length of pulses, every timing tick

When to latch (**TRLO_BIG_SCALER_LATCH_***xxx*):

LEADING_EDGE - leading edge
TRAILING_EDGE - trailing edge

SCALER(i)

Signal i to count.

SCALER_RESET

Reset scalers.

SCALER_LATCH(j)

Signal to reset generic scaler block j.

The latch inputs are common for blocks of TRLO_BIG_SCALER_LATCH_BLOCK scalers, i.e. scaler[0] to scaler[TRLO_BIG_SCALER_LATCH_BLOCK-1] is latched by the first latch input, and so on.

As output, the scaler delivers a 32-bit latched count value.

gen[i] 32-bit latched counter value.

The scalers can also be latched and reset via VME pulses:

pulse Action as per (**TRLO_BIG_PULSE_***xxx*):

SCALER_RESET - Reset general scalers.
SCALER_LATCH - Latch general scalers.

Even if the scalers can be reset, there is no need to reset the latched values - they will be new whenever latched. Furthermore, it is often easier to never reset scalers, but rather let them run continuously. Differences between two readings are easy to calculate in software.

MUX SOURCE SCALERS

Each multiplexer source (**TRLO_BIG_MUX_SRC_***xxx*) is also directly connected to a scaler, counting the number of leading edge pulses. They must be latched via VME before readout. They are intended for debugging or monitoring.

pulse Action as per (**TRLO_BIG_PULSE_***xxx*):

MUX_SRC_SCALER_RESET - Reset mux source scalers.
MUX_SRC_SCALER_LATCH - Latch mux source scalers.

As output, the scaler delivers 32-bit latched values.

mux_src[i]
32-bit latched counter value.

TIMING LATCHES

Latch the timing count on an event of the selected signal.

timer_latch_mode[i]
When to latch (**TRLO_BIG_TIMER_LATCH_MODE_***xxx*):

LEADING_EDGE - leading edge
TRAILING_EDGE - trailing edge

TIMER_LATCH
Signal to latch on.

The readout consists of two 32-bit values (lo and hi) of the latched timer. (In a previous version, there was a latch count in the 4 high bits (nibble) telling how many times the latch has latched. Possibly was useful to detect missed latches.)

timer_latch[i][2]
Output value, latched timing counter, lo and hi 32-bit word in [0] and [1].

MULTI-ENTRY TIMING LATCH

The time stamps (31 bits) of the timer latches are also recorded in individual multi-entry buffers (512 entries). If it becomes full (either due to slow read-out, or discard-on-full, see below), the high bit (32nd) of the data-word following lost entries is set to one.

60-bit time stamps can also be recorded in two consecutive words, starting with the low 30 bits. In this case, the next-highest bit (31st) is 0 for the low data words and 1 for the hi data words. When 31-bit timestamps are recorded, the 31st bit is part of the timestamp.

A control word allows to set an 'almost-full' level, at which an output at the multiplexer will go active. This can e.g. be connected to a pending trigger to enforce readout. A input from the multiplexer allows to select a discard-on-full mode where old entries are discarded thereby keeping the latest entries instead of the oldest. By some trickery with a edge-to-gate generator, this can be used to discard old entries until a trigger is accepted (preferably with some delay added), to ensure that data around the trigger is kept.

Reading from any address within the output array will provide the next value. A status word with the number of words (not triggers) left must be consulted before reading, as there is no unique no-valid-data word (0x5a5a5a5a will however be delivered).

multi_latch_control[i]
Almost-full level of the multi-entry buffer (**TRLO_BIG_MULTI_LATCH_CONTROL**_*xxx*):

ALM_FULL_LEVEL_MASK/SHIFT - Almost full level. Use ALM_FULL_LEVEL(n) to set value, with n='max' for safe value, or 'half'.

TWO_WORDS - Record 60-bit time stamps.

multi_latch_status[i]
Number of data words available (**TRLO_BIG_MULTI_LATCH_STATUS**_*xxx*):

DATA_AVAIL_MASK/SHIFT - Available words.

LOST_WRITE - Data lost due to full buffer. Set until a new entry is written in the buffer.

multi_latch[i][]
Next data word.

MULTI_LATCH_DISCARD_OLD(i)
Multiplexer input control signal.

MULTI_LATCH_ALM_FULL(i)
Multiplexer output control signal.

pulse Action as per (**TRLO_BIG_PULSE**_*xxx*):

MULTI_LATCH_CLEAR(i) - clear readout buffer i.

MULTI_LATCH_CLEAR_ALL - convenience bitmask to clear all buffers.

PATTERN LATCH (ALL MUX SOURCES)

Latch the values of all mux source signals on the leading edge of the selected signal, and store in output registers.

pattern_latch[i][j]
Latched bit-pattern with all mux sources. j denotes that several 32-bit registers are needed to cover the full mux source array.

PTN_LATCH(i)
Signal to latch on.

pulse VME-action as per (**TRLO_BIG_PULSE**_*xxx*):

PTN_LATCH(i) - latch into pattern i

PTN_LATCH_ALL - convenience bitmask of all pattern latches

MISCELLANEOUS

GENERAL CONTROL REGISTERS

timer_tick_period

Period in 10 ns steps of the slow timer counter. (16 bits). Default is 100, i.e. 1 us.

GENERAL OUTPUT REGISTERS

version_md5sum

md5sum of the full VHDL code. Available as (**TRLO_BIG_MD5SUM_***xxx*) constant in the C interface header file:

STAMP - 32 low bits of the stamp.

FULL - String with the entire md5sum.

compile_time

Time of the compile (seconds since 1970-01-01).

csr_parity[i]

Parity bits for all setup registers, 32 in each word.

timing_tick[2]

Current value of the full-speed 64-bit timing counter, lo and hi 32-bit word in [0] and [1], latched via VME (TRLO_BIG_PULSE_TIMER_LATCH).

deadtime_tick[2]

Count of full-speed timing ticks when deadtime (internal inhibit) was active. (c.f. the 'timing_tick' output register.) Not latched, if using the high word, make sure it did not change while reading the low.

last_dt_release[2]

Time of last deadtime release, after non-0 triggers (i.e. not pure multi-event-triggers). (c.f. the 'timing_tick' output register.)

GENERAL PULSE REGISTERS

pulse Action as per (**TRLO_BIG_PULSE_***xxx*):

TIMER_RESET - Reset the full-speed timing counter. Note that due to implementation details, if a timer channel has been recently latched (up to including this cycle), the latched value will be negative. Recently is a few ten clock cycles (precisely r=2*2^n, where 2^n >= the total number of timing latches). The reset is only expected to be used on startup, and most often not even then.

TIMER_LATCH - Latch the full-speed timing counter.

FIXED SIGNALS

WIRED_ZERO
Fixed signal 0.

WIRED_ONE
Fixed signal 1.

FRONT-PANEL LEDS

The front-panel LEDs can show the status of any signal from the multiplexer. The last two LEDs are hard-wired to show if the serial timestamp and the triva mimic have a lock on their respective serial input bit-streams.

FRONT_LED(i)
Signal to LED (i).

SERIAL TIMESTAMP DISTRIBUTION

The current value of the local full-speed timing counter can be sent via a serial protocol to other modules. The receiving ends automatically synchronise to the signal and store time-stamps when receiving independent latch signals. The precision of the latched time-stamps only depend on the clock frequency of the receiving module, with a sigma of about 0.35 clock cycles and worst deviation of 2 clock cycles observed during testing. The serial signal can be transported using 'any' cable means. The top front-panel LED on a VULOM module or second bottom on a TRIDI module indicates receiver protocol lock.

Each serial message super-cycle consist of 16 data-cycles of 32 symbols of synchronisation pattern and 32 symbols with data payload. The payload is Hamming encoded to enable forward error correction, with a total of 11 payload bits and 5 parity bits. 4 bits are used for the time information and 7 for a set of multiplexed signals. The synchronisation pattern is also used to monitor the signal integrity by counting bad symbols. (The synchronisation pattern is carefully crafted such that there is a margin of 2 symbols until it is equally possible that a suspected wrongly received synchronisation in fact is any other possible pattern sequence including encoded data.)

The serial protocol is designed such that exactly half of the signal symbols are 0 and the other half 1. This is achieved by Manchester encoding of the data and a balanced synchronisation pattern. There are at most two symbols with the same value next to each other; sequences of such pairs are delivered by the synchronisation pattern, as well as sequences of fast-flipping symbols. Both these are used by the receiver when doing its initial frequency search (i.e. symbol length determination), which may take about one second.

After rough frequency lock, the receiver goes into a phase-tracking mode using the 0-1 transitions with slow frequency correction. Symbols are sampled at the middle of each slot. Once the receiver has locked onto the bit-stream, it will stay locked even if a number of transitions should be missing. The phase-tracking essentially works by employing a fractional counter such that it can predict when the next 0-1 transition should occur. Each actual transition leads to a correction of half receiver clock cycle. When a counter has seen 4 more corrections in one direction, the frequency is adjusted slightly. When the frequency estimate is spot-on, the phase will just correct slightly forth-and-back.

The result of the phase tracking is used to provide the low part of the received time whenever a latch is requested. The high part is provided by the time information received every message super-cycle. Time information in adjacent message super-cycles must match for the timing receiver to consider time as good. Super-cycles with known transmission errors are overcome by the phase track counting.

The multi-entry timestamp buffer has 512 entries and thus can store up to 256 entries. If it becomes full,

and a timestamp thereby cannot be stored and is lost, the following stored timestamp will have the high bit of the second word (time hi) set to one. If the timestamp receiver is desyncronised, the second highest bit of the second word (time hi) is set to one.

SENDER HANDLING

serial_timestamp_speed

Symbol-length of the serial message. Period = 8 * 2^n, i.e. 8, 16, 32 or 64. Note that time slewing (see below) affects the period.

For the receiver to lock and track the signal, the SERIAL_TSTAMP_OUT count rate should roughly be in the range between 350000 to 9500000 pulses/s.

serial_timestamp_aux_bits

Set the auxiliary bits (and or in signals).

SERIAL_TSTAMP_OUT

Serial message output from sender.

RECEIVER HANDLING

serial_timestamp_buf_control

Almost-full level of the multi-trigger buffer (**TRLO_BIG_MULTI_TRIG_BUF_CONTROL**_*xxx*):

ALM_FULL_LEVEL_MASK/SHIFT - Almost full level.

serial_timestamp_status

Number of data words available in bits 0-9. Bit 15 marks desynchronised serial message reception. Bit 16 marks bitstream sync, and bit 17 that bitstream has had sync loss. Bit 18 marks data pattern sync, and bit 19 that the data pattern has had sync loss. Bits 20-23 is a counter of bad bits, and bits 24-31 is a checksum (XOR) of the data currently in the output buffer.

Bits 17, 19 and 20-23 are cleared by a pulse, see below.

serial_tstamp[]

Next data word from multi-timestamp buffer.

SERIAL_TSTAMP_IN

Serial message input to receiver (decoder).

SERIAL_TSTAMP_LATCH

Latch a timestamp. (Creates 2-word entry in buffer).

SERIAL_TSTAMP_ALM_FULL

Timestamp output buffer is almost full.

SERIAL_TSTAMP_DESYNC
Signal marking reception desynchronisaition.

pulse Action as per (**TRLO_BIG_PULSE**_*xxx*):

SERIAL_TSTAMP_BUF_CLEAR - Clear the latched timestamp buffer.

SERIAL_TSTAMP_FAIL_CLEAR - Clear the flip-flops remembering reception desynchronisation and bad bits.

MULTIPLEXED SIGNALS

Seven signals can also be multiplexed within the serial message. They are delivered every data-cycle and updated in the multiplexer of the receivers. In case the receiver looses lock of the serial message, the output signals will be output as 0.

15 bits of auxiliary data bits are also transmitted along with the time. They can be used to transport some information from the sender to the receievers, or e.g. be used to identify the sending stream.

The signals are introduced by the sender:

SERIAL_SIGNALS_IN(i)
Multiplexed signal i.

serial_timestamp_aux_bits
Auxiliary data bits to send, in bits 0-14. Bits 16-22 are or'ed together with the multiplexed signals.

Available at the receiving end:

SERIAL_SIGNALS_OUT(i)
Multiplexed signal i.

serial_timestamp_aux_status
Value of auxiliary data bits in low bits 0-14, at last end of super-cycle. Bits 16-22 contain the serial signals, at last end of data-cycle. Bit 23 marks desynchronised serial message reception.

HEIMTIME (SPEAKING CLOCK) SENDER

In order to rather easily share a common time reference with foreign DAQ systems that have no means to use e.g. the serial time protocol, a simple "speaking clock" protocol is implemented.

Provided that the foreign DAQ system is able to locally timestamp a received logical signal, it can receive the periodic signals of the heimtime protocol, and during analysis the common time scale (as provided by the TRLO_BIG II) can be recovered.

The protocol consists of two parts. Every 2^19 local clock cycles a pulse is generated. With the local clock of 100 MHz this means every 5.24288 ms (or 190.7 Hz of signals). In order to tell time, for 32 pulses starting every 2^26 ticks (or 128 pulses, or about 0.671 s apart), it delivers two additional pulses. They either have a separation of 0.16384 or 0.65536 ms. The short separation means 0, and the long separation means

1, in a 32-bit time stamp. The 32-bit time-stamp starts at local bit 24.

For analysis, reception of one full time message would be enough, as it then can perform dead counting of the pulses. It is naturally recommended to continuously verify that the received timestamps match with the previous ones.

The reason for having both this and the serial timestamp protocol is that the serial protocol lends itself to easy FPGA decoding and precision following, while this Heimtime protocol allows for rather straightforward handling in analysis, without requiring tremendous amounts of data to be recorded by the foreign DAQ system.

SENDER HANDLING

HEIMTIME_OUT
Heimtime protocol output.

TIME SLEWING

In order to be able to precisely track another time scale, the speed and offset of the local source of the serial and Heimtime protocols can be adjusted. A user-specified value is added to the slewed time counter every clock cycle. Note that the 24 low bits are added below the value being sent. They provide precision control as their constribution accumulated over many clock cycles. The absolute value of the counter can also be shifted by setting and the adding an offset.

slew_counter_add
Value to add to the slewed time counter every clock cycle.

Use the value 0x1000000 to have the nominal serial timestamp sender bit slot periods.

slew_counter_offset
Value to add to the slew time counter on a pulse.

slew_counter
Action pulse to add the offset, as per (**TRLO_BIG_SLEW_COUNTER_***xxx*):

ADD_OFFSET_LO - Add to low 32 (real) bits.

ADD_OFFSET_HI - Add to high 32 bits.

SLEW_LATCH
Latch the current value of the slewed time counter.

slew_counter_cur_lo
Low 32 bits of the latched time counter.

slew_counter_cur_hi
High 32 bits of the latched time counter.

TRIMI (TRIVA MIMIC)

When used with the MBS, the encoded trigger from the trigger state machine is connected to the master TRIVA module trigger inputs, and the deadtime from the TRIVA sent back to the trigger state machine.

The task of the TRIVA is to notify the controlling processor of each readout event and keep deadtime until the readout process has requested its release. The notification can be either by interrupt or by register polling by the readout program.

In a multi-branch setup, the TRIVA module of each slave crate also form the interface with the local readout processors. The trigger bus between the TRIVA modules deliver the triggers from the master system and collect the deadtime from all slaves. It also ensures synchronous event-wise operation of the entire multi-branch system.

The TRIMI part of the TRLO_BIG II performs the same function as a TRIVA, but within the same module. It has the same register layout, making it immediately compatible with the MBS data acquisition system.

For a multi-branch system, the TRIMI uses a unidirectional serial protocol to distribute triggers and deadtime return by simple on/off signalling: serial trigger link with common deadtime (STL/CDT). The receiving (slave) end automatically synchronise to the (STL) signal. The bottom front-panel LED on a VULOM or TRIDI module module indicate receiver protocol lock. The serial link-signal can be transported using 'any' cable means. It may be fanned out directly at the master module, or in a tree-like fashion at slave or other modules. Likewise, the deadtime return signal may be or'ed at the TRIMI module or earlier points. When the TRIMI module inputs are used to collect the deadtime signals, some specialised monitoring facilities can simplify debugging.

For larger systems, the serial trigger link can be arranged in a tree-like fashion. If a TRIMI module of an intermediate node is used for STL fan-out and CDT fan-in, it can do individual monitoring of the deadtime from the subsystems. It can also be reconfigured as a local master without any recabling. Independently of how the STL is organised as a tree in one or several layers, the MBS can still treat the system as a flat topology.

The typical failure mode (trigger desynchronisation) of a multi-branch TRIMI system is either that the STL is being run with a too short symbol period or the deadtime is improperly wired. Both cases are detected by the slave systems, which will notice the missing event counters and report trigger mismatch.

The serial protocol is similar to the one used by the serial timestamp distribution. A 32-bit idle pattern is sent continuously, which serves both as help during setup, and allows the receiver to lock onto the signal. At every second symbol, the transmitter may send a special 18-symbol start pattern, indicating that a trigger will follow. The trigger payload is a 4 bit trigger number, 4 bit event counter and 2 bits of an multi-event counter. It is Hamming encoded to enable forward error correction, leading to a total of 15 bits or 30 symbols due to the Manchester encoding of the data. Trigger link reset commands, as well as side-band 8-bit messages are transmitted in the same way, but with a different start pattern.

Trigger latency is 18 + 30 symbol periods. With a minimum receivable symbol period of 6.75 local clock cycles, usually chosen as 10, this at a transmitter clock of 100 MHz corresponds to 4.8 us. While not fully implemented yet, the system is prepared to allow the slave systems to issue the interrupt to the processor after the header word, i.e. 1.8 us. With long connection distances, signal dispersion may require longer symbol lengths. An alternative is to use optical transceivers, which also prevent grounding issues.

INPUT SIGNALS

TRIMI_TDT

Deadtime from the TRIMI.

TRIVA-COMPATIBLE REGISTERS

status Status register, trigger and event counter.

control Setup register.

fcatime
Fast clear time.

ctime Conversion time. Delay before processor notification.

TRIMI REGISTERS

link_status
STL/CDT receiver status.

link_control
Control register.

link_sendmsg
Soft message send (write) and last received (read).

link_speed
Symbol width in local clock cycles (+5).

link_fast_dttime
Fast deadtime (cover time until deadtime from slaves arrive).

TRIMI CONNECTIONS

link_serial_in
Select front-panel for STL input signal.

trig_in Select front-panel input for encoded trigger. Uses selected and next three inputs; selected input must be a multiple of four.

Use special value to use the encoded trigger from the TRLO_BIG II.

Output signals may be directed to any front-panel output i, set by the bitmask bit (i%16) in conn_out[i/16]:

link STL output signal bitmask. A direct copy of the STL input signal if the TRIMI is in slave mode.

dt Deadtime output bitmask.

Deadtime from slaves can be received from any front-panel input i, set by the bitmask bit (i%16) in dt_in[i/16]:

enable Enable bitmask.

advisory
Enable bitmask for advisory deadtimes. These are not subject to the checking for missing or double-pulsing. Useful for deadtimes from time-stamped 'slave' systems, which legally both may ignore (miss) triggers, or issue their own additional local triggers.

DEADTIME RECEIVER

Further entries in dt_in[i/16] can be used to diagnose the deadtime return signals:

current
Current status bitmask of deadtime. Note: all inputs are shown, regardless of the enable/advisory bitmask. To allow check of not-yet connected (enabled) systems.

good Input has delivered deadtime for this trigger before the fast slave deadtime elapsed. For all inputs, regardless of the enable bitmask. (Reset each trigger).

dbl Input has delivered two deadtime signals between between two successive fast slave deadtime expire points.

DEADTIME MONITOR

The start and end of the global deadtime, as well as the end of the local deadtime, and the end of all enabled deadtime receiver inputs are recorded in a 512-entry buffer.

dt_mon_status
Number of words available in the deadtime monitor buffer.

trimi_dt_mon[]
Next data word from the deadtime monitor.

FRONT-PANEL DISPLAY

The front-panel 24x36 LCD-display is used to show information about the status of the fast_path and the trigger state-machine. Note that this information also is available over VME, in many cases also as scalers for much better rate-estimates.

```
X  Dd.  Bb.  lo.  I.   Y Z
X                    Y Z
X  TM   A A  st   re   Y Z
X       A A            Y Z
X  A.   A A  VM   VK
```

Where the **X-column**
read from below, is double-dots showing active fast_path LMU inputs, the top four mark the AUX LMU inputs.

The **Y-column**
from below, show the active LMU outputs.

The **Z-column**
from below, show accepted TPAT bits, i.e. after reduction.

The two **A-columns**
from below, show accepted triggers, 0-7 in the left column and 8-15 in the right.

Dd. marks (D) DT from triva input, (d) internal DT flip-flop, (i) internal inhibit (internal fast deadtime), (X) when active LMU output prevents trigger state machine to become IDLE, or (.) for none.

Bb. marks (B) busy input, (b) internal busy flip-flop, or (.) if none.

lo. marks the (1-based hexadecimal) number of the first stuck LMU output that is enabled, or (.) if none. See lmu_enabled_stuck_out.

I. marks inhibit (trigger state-machine veto) (I) or (.) for none. A skull is shown when active LMU outputs prevent normal triggering.

A. shows the accepted trigger number in hexadecimal (.) if none.

re shows the reason for the current trigger, i.e. which path it took in the state-machine.

st shows the trigger state machine state. (1) is IDLE, (B=11) is WAIT_TRIVA, (C=12) is TRIVA_DONE (i.e. wait for busy release).

VM and VK
are the high byte of the VME address, i.e. module setting. Above these, two rows of bits show the 20 lowest bits of the trigger counter. The leftmost bit of the lower row flips back-and-forth every 2^10 = 1024 events, and the leftmost bit of the upper row every 2^20 events. These bits blinking show if events are being processed and give an idea of the rate.

TM triva mimic info: (H) if not in go mode, otherwise the current trigger number when in (global) deadtime or (.) if idle. Active local deadtime is shown as a line below the trigger number. To the right, four markers from top to bottom: locked on serial trigger bus signal, slave mode, irq pending and mismatch.

EXAMPLES OF THE GENERICS USAGE

LAND SPECIALITIES

The TCAL and CLOCK triggers are generated by pending triggers, in turn provided by pulsers from the general logics. Different rates in- and off-spill can be selected by using the general LMU. Thus, these triggers no longer eat TPAT/trigger-box entries. By using the pending instead of pulsed trigger input, rates can be guaranteed.

Spill-mimic from the accelerator BOS and EOS signals is handled by the general edge-to-gate conversion. The in/off-spill coincidence signal is provided by the aux input to the fast-path LMU, thereby not consuming a separate input for spill-is-on. BOS and EOS triggers are handled by the pending trigger system to

guarantee delivery.

(Post) pile-up rejection is supported by using the general stretcher which can provide the pile-up veto signal to an auxiliary fast-path LMU input.

Monitoring of the per-event begin and end of dead-time is provided by the timer-latches. Can also be measured for individual systems with the TRIDIs.

(REX)-ISOLDE SPECIALITIES

Multi-event mode is used to make the most of the awkward (REX)-ISOLDE duty cycle.

Clock latches are used to record the times of T1 and T2 (protons on target and REX-pulse, respectively). It is no longer necessary to make a trigger for these events.

(If one still want to make an EBIS trigger, the delay generators can be used to delay T2 to make the trigger happen after the REX pulse, so that the read-out does not occur during the ion burst.)