Command Interface Specification for the ACC / ACDC Cards

Author: David Greenshields

Date: 16 Jun 2021

ACC firmware version: V301

ACDC firmware version: V302

# Introduction

The ACC card communicates with the control computer via a USB link. The ACC is connected to between 1 and 8 ACDC cards and has a dedicated LVDS connection to each of them. Communication is done via these LVDS lines using a synchronous coding protocol running at 40Mbps.

Commands sent to the ACC over usb are 32-bit words. The 32-bit word is split into fields for acdc board select (bit 31:24) and command type (bit 23:20). Command type determines whether the command is for the ACC or for the ACDC.

To send a command to the ACC, the top 8 bits should be zero and the next 4 bits will be in the range 0 to 9, indicating the command type.

To send a command to the ACDC board (via the ACC) the top 8 bits select which board(s) the command should be sent to and the next 4 bits select the command type in the range A to F (hex).

A FIFO buffers commands destined for the ACDC ready to be sent over the serial link.

### Data Framing

Communication between ACC and ACDC is done using 8b10b coding where each byte is coded into a 10-bit word. When no data is to be sent a special line idle code is sent instead. Every millisecond a 2-byte sync word is sent so that the receiver can lock on if not locked, or if already locked it can perform a timeout to check that the sync words are being received regularly.

All data transmitted serially and on the usb are grouped in pairs to form 16-bit words.

The ACDC can send 3 different types of frame to the ACC:

1. ID frame: 32 words, containing version info etc.
2. PPS frame: 16 words, containing a timestamp of the pps (pulse per second) trigger
3. PSEC frame: 7795 words, containing data from all the PSEC chip, plus other info.

*Note: pps frames and psec frames are sent automatically when in the appropriate mode. ID frames are requested by the control computer.*

The ACC can send 2 different types of frame to the control computer:

1. ID frame: 32 words with version info and local setup/status data info.
2. rx buffer frame: Variable length- transmits the contents of the serial receive buffer (for the selected channel). This data is sent unaltered i.e. no additional info is added.

The above frame types are requested by the control computer. Flag bits and data fields in the ID frame show whether an rx buffer on any channel has received a frame from the ACDC, and what length it is.

# 32-bit Word Format

[31:24] = ACDC Board select mask (1 = selected, 0 = not selected)

[23:20] = Command type (0-9 = ACC command, A-F = ACDC command)

[19:16] = option

[15:0] = value

The above is the general format although some commands differ slightly below bit 20 in order to accommodate specific bit fields.

# ACC Commands

## 0x00000000 Global reset request

Performs a hardware reset on the ACC card. (ACDCs not affected)

## 0x000200XX RX buffer reset request

Clears the contents of the selected uart receive buffers , making them ready to receive another frame from the ACDC board. (8 bits)

## 0x00100000 Software trigger

Generates a trigger pulse to all acdc cards that have been set up for software trigger mode.

## 0x00200000 Local info read request

Prompts the ACC to send a short frame (32 words) back to the control computer containing information about the ACC board setup and status. (See later sections in this document for details)

## 0x0021000N Serial buffer read request

Causes the ACC to send the contents of the serial receive buffer of channel N (0 to 7) to the control computer, i.e. the frame received from the ACDC.

## 0x00300XXM Trigger mode

Set trigger mode to value M (0 to 9) for the ACDC boards selected by XX [7:0].

Each channel can have its own setting. Trigger modes are:

1. Off
2. Software trigger
3. SMA trigger (ACC)
4. SMA trigger (ACDC)
5. Self-trigger
6. Self-trigger with SMA validation (ACC) & pps multiplexed trig
7. Self-trigger with SMA validation (ACDC)
8. SMA trigger (ACC) with SMA validation (ACDC)
9. SMA trigger (ACDC) with SMA validation (ACC) & pps multiplexed trig
10. PPS trigger (ACC pulse-per-second signal)

For modes using SMA validation from the ACC, an option exists to multiplex the pps triggers onto the validation line so that when valid is low pps trigger is enabled.

## 0x0031000N SMA Invert

Applies to the ACC SMA connector used for beam gate or ext trigger input.

N = 0 The SMA input is normal

N = 1 The SMA input is inverted

## 0x0032NNNN Beam Gate Window Start

Beam Gate window start delay from beam gate trigger.

time = 25ns x N (16 bit word, unsigned)

## 0x0033NNNN Beam Gate Window Length

time = 25ns x N (16 bit word, unsigned)

## 0x0034NNNN PPS Divide Ratio

The value N (16 bit word, unsigned) specifies the divider to be applied to the PPS signal.

N = 0 pps disabled

N > 0 trigger pulse every N seconds

## 0x0035000N PPS / Beam Gate Multiplexer

Applies to modes using beam gate validation via ACC (modes 5 and 8).

When enabled allows pps triggers multiplexed with signal trigger using beam gate window.

N = 0 disabled

N = 1 enabled

## 0x004NXXXX LED mode select

N = led select [0= green, 1=yellow, 2=red]

XXXX =

[7:0] signal source select [see firmware ‘acc\_main.vhd’]

[11:8] channel select

[12] invert

[13] flash

[15:14] *not used*

## 0x004F000X LED preset select

Set all leds to a preset setup.

X=0: default [usb tx/rx, use ext ref, board 1 detect]

X= 1: serial error check

X=2: sma input monitor [3:1]

X=3: sma input monitor [6:4]

X=4: System in & pps monitor

plus others... see firmware acc\_main.vhd

## 0x009X000N Test commands

0090000N N=1 use SMA3 for pps input; N=0 normal (lvds i/p)

0091000N N=1 use SMA4 for beamgate trigger input; N=0 normal (lvds i/p)

# ACDC Commands

*Note: The commands shown here have the board mask set to ‘FF’ but this should set as required depending on which boards the command is to be sent to.*

## 0xFFA00000 Set DLL VDD

Set a new value for analogue parameter ‘DLL VDD’ on the selected PSEC4 chips.

The bit fields are:

[16:12] PSEC4 mask – selects which chips the command will apply to. (5 bits)

[11:0] The new value (12 bits)

## 0xFFA20000 Set pedestal offset

Set a new value for pedestal offset (analogue input dc bias) on the selected PSEC4 chips.

The bit fields are:

[16:12] PSEC4 mask – selects which chips the command will apply to. (5 bits)

[11:0] The new value (12 bits)

## 0xFFA40000 Set ring oscillator control voltage

Set a new value for analogue parameter ‘ring oscillator control voltage’. This is a voltage which controls the delay and hence operating frequency of the DLL. It is used in the Wilkinson feedback loop to stabilize the sampling frequency to the required value.

The bit fields are:

[16:12] PSEC4 mask – selects which chips the command will apply to. (5 bits)

[11:0] The new value (12 bits)

## 0xFFA60000 Set self-trigger threshold

Set a new value for the self-trigger threshold voltage. This voltage determines the level of input signal that will cause a self-trigger signal to be generated. The bit fields are:

[16:12] PSEC4 mask – selects which chips the command will apply to. (5 bits)

[11:0] The new value (12 bits)

## 0xFFB0000M Set trigger mode

Set trigger mode to value M (0 to 9). See command 0x0030000M above for a list of trigger modes.

## 0xFFB10000 Self trigger mask 0

Set self-trigger enable bits for each channel of PSEC4 device 0. (1=enable, 0=disable)

bit field [5:0]. (6 bits in total)

## 0xFFB11000 Self trigger mask 1

Set self-trigger enable bits for each channel of PSEC4 device 1. (1=enable, 0=disable)

bit field [5:0]. (6 bits in total)

## 0xFFB12000 Self trigger mask 2

Set self-trigger enable bits for each channel of PSEC4 device 2. (1=enable, 0=disable)

bit field [5:0]. (6 bits in total)

## 0xFFB13000 Self trigger mask 3

Set self-trigger enable bits for each channel of PSEC4 device 3. (1=enable, 0=disable)

bit field [5:0]. (6 bits in total)

## 0xFFB14000 Self trigger mask 4

Set self-trigger enable bits for each channel of PSEC4 device 4. (1=enable, 0=disable)

bit field [5:0]. (6 bits in total)

## 0xFFB150NN Self trigger coincidence min

Set the minimum number of channels out of all 30 channels that must be high in order to create a self-trigger event.

NN = value 0 to 30

## 0xFFB1600N Self trigger sign

Set the polarity of the self-trigger voltage comparator.

N = value 0 or 1

## 0xFFB1800N Self trigger use coincidence

N = value 0 or 1

0 = coincidence not used. Self-trigger is generated by OR-ing all enabled self-trig channels.

1 = coincidence used. Self-trig generated when coincidence level is above min value.

## 0xFFB2000N ACDC SMA invert

N = value 0 or 1

0 = normal polarity (high level or rising edge)

1 = inverted polarity (low level or falling edge)

*Note: Applies to the ACDC SMA connector when used as trigger or as validation input*

## 0xFFB50000 Transfer enable request

This command sets the ‘transfer enable’ signal high, meaning that the ACDC may transfer one frame of data to the ACC at any time. Once a transfer has been done, transfer enable automatically goes low.

This command should be sent whenever the data from the uart rx buffer has been read. Then it signals to the ACDC that the rx buffer is empty and is ready to receive data.

## 0xFFB51000 Trigger reset request

Resets the trigger state machine.

Should not be needed in normal operation.

## 0xFFB52000 Event and time reset request

Resets the trigger event counter to zero and resets the 64-bit system time counter to zero.

## 0xFFB54000 PSEC4 frame transfer disable request

This command sets the ‘transfer enable’ signal low, meaning that the ACDC is not allowed to transfer data to the ACC.

If this command is sent just after a transfer has already been started, the transfer will continue to the end and the command will have no effect.

The best thing is to apply a suitable delay after sending this command to ensure that any frame which was in progress is now finished and no more PSEC4 frames will be sent.

This command can be used before requesting an ID data frame from the ACDC so that PSEC4 frames are suppressed and do not interfere with the ID frame transmission.

## 0xFFB6000N Trigger test mode – option 0: no transfer

N = 1: a trigger event will not cause a transfer of data to the ACDC, nor will it wait for the transfer enable signal. This part of the process will be skipped and the trigger state machine will reset ready for the next trigger.

A trigger event will still be recorded so this mode is useful for testing correct operation of trigger signals and modes.

N = 0: normal operation.

## 0xFFC0XXXX Calibration enable

Set the calibration enable bits [14:0] (15 bits total). Each bit operates 2 channels of a PSEC4 chip. The channels are paired as {0,1} {2,3} {4,5} for each chip.

## 0xFFD00000 ID frame request

Requests a 32-word ID frame to be sent from the ACDC to ACC.

*Note: This is not dependent on signal ‘transfer enable’ which only applies to PSEC4 frames*

See later sections of this document for frame details.

Note that PSEC4 frames should be disabled first using command 0xFFB54000 to prevent conflicts.

## 0xFFF0000N Test mode 0: Use sequenced PSEC4 data

N = 1: uses data values of 0 to 1535 for the data instead of real PSEC4 data.

N = 0: normal operation.

## 0xFFF20000 DLL reset request

Resets the delay-locked-loop of all the PSEC4 devices.

## 0xFFFF0000 Global reset request

Hardware reset of the ACDC board.

## Board Detection

When the system first powers up, the acc needs to determine which of its 8 ports are connected to working ACDC cards. This is done by reading the 32 word ID frame from the ACC and then looking at bits 7:0 of word 7. A ‘1’in the corresponding position says that the acc serial rx processor is locked to the received signal, therefore a board is present.

## Read ACDC version info

Send 0xFFB54000 // disable psec4 frame transfer to ACC

Wait 100ms // Wait for any psec4 frames to finish – these would interfere with ID frame

Send 0x000200FF // reset all of the uart receive buffers

Send 0xFFD00000 // request a 32-word ID frame from all acdc cards

Wait 100ms // wait to allow for frame transfer (ACDC => ACC)

Send 0x00200000 // read the ACC info frame. Look at word 7, bits [7:0] for board detect info

// 1=detected, 0=not detected

// Also check the uart rx buffer length is equal to 32 words for all active

// channels (words 16 to 23 in the frame data)

## Trigger Setup

### Trigger Mode

The trigger mode options are very flexible, and each ACDC card can have its own trigger mode.

The simplest procedure for setting trigger mode N for all boards is:

Send 0x00300FFN //ACC trig setup: Set trigger mode N for all channels (ACDC cards)

Send 0xFFB0000N //ACDC trig setup: Set trigger mode N for all channels (ACDC cards)

Note that both ACC and ACDC must be set up with the same mode number – otherwise unpredictable results will occur.

In addition, to fully set up the trigger arrangements, the following may need to be adjusted accordingly:

### Self-trigger

send 0xFFB100NN //set up self trigger mask

send 0xFFB110NN //set up self trigger mask

send 0xFFB120NN //set up self trigger mask

send 0xFFB130NN //set up self trigger mask

send 0xFFB140NN //set up self trigger mask

send 0xFFB150NN //set up self trigger coincidence minimum

send 0xFFB160NN //set up self trigger sign

send 0xFFB170NN //set up self trigger detection mode

send 0xFFB180NN //set up self trigger use coincidence

### ACDC SMA

send 0xFFB200NN //set up ACDC SMA invert

### Trigger validation

send 0x0032NNNN //beamgate window start delay

send 0x0033NNNN //beamgate window length

## Data Acquisition

General procedure is as follows:

1. Initialize:
   1. Check which ACDC boards are present
   2. Set up the trigger as required
   3. Clear serial receive buffers
2. Prepare for trigger event:
   1. Enable psec4 data frame transfer
3. Wait for data:
   1. Request ACC local info frame (32 words)
   2. Check if frame received for each channel, if so record data
4. Record the data:
   1. Check the rx data length by looking at the acc short frame
   2. Request a read of ACC serial rx buffer on the corresponding channel
   3. Read the data
   4. When all channels done, loop back to step 2

*Note: serial receive buffers automatically reset when the data is read out.*

# ACC ID Frame Details

localData(0) <= x"1234";

localData(1) <= x"AAAA";

localData(2) <= firwareVersion.number;

localData(3) <= firwareVersion.year;

localData(4) <= firwareVersion.MMDD;

localData(5) <= x"0000";

localData(6) <= x"0000";

localData(7) <= x"00" & acdcBoardDetect;

localData(8) <= x"000" & "00" & trig.ppsMux\_enable & trig.SMA\_invert;

localData(9) <= x"0000";

localData(10) <= x"0000";

localData(11) <= x"0000";

localData(12) <= x"000" & "00" & pllLock & useExtRef;

localData(13) <= x"0000";

localData(14) <= x"00" & frame\_received;

localData(15) <= x"0000";

localData(16) <= std\_logic\_vector(to\_unsigned(rxDataLen(0),16));

localData(17) <= std\_logic\_vector(to\_unsigned(rxDataLen(1),16));

localData(18) <= std\_logic\_vector(to\_unsigned(rxDataLen(2),16));

localData(19) <= std\_logic\_vector(to\_unsigned(rxDataLen(3),16));

localData(20) <= std\_logic\_vector(to\_unsigned(rxDataLen(4),16));

localData(21) <= std\_logic\_vector(to\_unsigned(rxDataLen(5),16));

localData(22) <= std\_logic\_vector(to\_unsigned(rxDataLen(6),16));

localData(23) <= std\_logic\_vector(to\_unsigned(rxDataLen(7),16));

localData(24) <= serialRx\_statusWord(31 downto 16);

localData(25) <= serialRx\_statusWord(15 downto 0);

localData(26) <= x"0000";

localData(27) <= x"0000";

localData(28) <= x"0000";

localData(29) <= x"0000";

localData(30) <= x"AAAA";

localData(31) <= x"4321";

*for i in 0 to 7 loop*

*serialRx\_statusWord(i\*4 + 0) <= serialRx.symbol\_align\_error(i);*

*serialRx\_statusWord(i\*4 + 1) <= serialRx.rx\_clock\_fail(i);*

*serialRx\_statusWord(i\*4 + 2) <= serialRx.disparity\_error(i);*

*serialRx\_statusWord(i\*4 + 3) <= serialRx.symbol\_code\_error(i);*

*end loop;*

# ACDC ID frame details

IDframe\_data(0) <= x"1234";

IDframe\_data(1) <= x"BBBB";

IDframe\_data(2) <= firwareVersion.number;

IDframe\_data(3) <= firwareVersion.year;

IDframe\_data(4) <= firwareVersion.MMDD;

IDframe\_data(5) <= x"000" & "000" & serialRx.disparity\_error;

IDframe\_data(6) <= x"0000";

IDframe\_data(7) <= x"0000";

IDframe\_data(8) <= x"0000";

IDframe\_data(9) <= info(0,1); -- wlkn feedback current (channel 0)

IDframe\_data(10) <= info(0,2); -- wlkn feedback target (channel 0)

IDframe\_data(11) <= std\_logic\_vector(to\_unsigned(ppsCount,32))(31 downto 16);

IDframe\_data(12) <= std\_logic\_vector(to\_unsigned(ppsCount,32))(15 downto 0);

IDframe\_data(13) <= std\_logic\_vector(to\_unsigned(beamGateCount,32))(31 downto 16);

IDframe\_data(14) <= std\_logic\_vector(to\_unsigned(beamGateCount,32))(15 downto 0);

IDframe\_data(15) <= std\_logic\_vector(to\_unsigned(eventCount,32))(31 downto 16);

IDframe\_data(16) <= std\_logic\_vector(to\_unsigned(eventCount,32))(15 downto 0);

IDframe\_data(17) <= std\_logic\_vector(to\_unsigned(IDframeCount,32))(31 downto 16);

IDframe\_data(18) <= std\_logic\_vector(to\_unsigned(IDframeCount,32))(15 downto 0);

IDframe\_data(19) <= x"0000";

IDframe\_data(20) <= x"0000";

IDframe\_data(21) <= x"0000";

IDframe\_data(22) <= x"0000";

IDframe\_data(23) <= x"0000";

IDframe\_data(24) <= x"0000";

IDframe\_data(25) <= x"0000";

IDframe\_data(26) <= x"0000";

IDframe\_data(27) <= x"0000";

IDframe\_data(28) <= std\_logic\_vector(to\_unsigned(serialNumber,32))(31 downto 16);

IDframe\_data(29) <= std\_logic\_vector(to\_unsigned(serialNumber,32))(15 downto 0);

IDframe\_data(30) <= x"BBBB";

IDframe\_data(31) <= x"4321";

*Notes:*

serial number = number of frames sent

# ACDC PPS frame details

when 0 => txWord := x"1234";

when 1 => txWord := x"EEEE";

when 2 => txWord := timestamp(63 downto 48);

when 3 => txWord := timestamp(47 downto 32);

when 4 => txWord := timestamp(31 downto 16);

when 5 => txWord := timestamp(15 downto 0);

when 6 => txWord := std\_logic\_vector(to\_unsigned(serialNumber,32))(31 downto 16);

when 7 => txWord := std\_logic\_vector(to\_unsigned(serialNumber,32))(15 downto 0);

when 8 => txWord := std\_logic\_vector(to\_unsigned(ppsCount,32))(31 downto 16);

when 9 => txWord := std\_logic\_vector(to\_unsigned(ppsCount,32))(15 downto 0);

when 10 => txWord := x"0000";

when 11 => txWord := x"0000";

when 12 => txWord := x"0000";

when 13 => txWord := x"0000";

when 14 => txWord := x"EEEE";

when 15 => txWord := x"4321";

# PSEC Data Frame Structure

## Once per frame:

0x1234 Startword

0xA5EC Frame type ID

## Once per PSEC 4 chip (5 in total):

0xF005 Preamble

Data x 256 Channel 0 data

Data x 256 Channel 1 data

Data x 256 Channel 2 data

Data x 256 Channel 3 data

Data x 256 Channel 4 data

Data x 256 Channel 5 data

Info 0 to 13 Information words – see below

0xFACE PSEC end word

## Once per frame:

Trig rate x 6 PSEC0 Self trig rate counts (trig events per 1sec interval for each of the 6 channels)

Trig rate x 6 PSEC1 Self trig rate counts (trig events per 1sec interval for each of the 6 channels)

Trig rate x 6 PSEC2 Self trig rate counts (trig events per 1sec interval for each of the 6 channels)

Trig rate x 6 PSEC3 Self trig rate counts (trig events per 1sec interval for each of the 6 channels)

Trig rate x 6 PSEC4 Self trig rate counts (trig events per 1sec interval for each of the 6 channels)

Trig rate Combined trigger rate count (trig events per 1 sec interval)

0xA5EC Frame ID

0x4321 Endword

## Total: 2 + (5 \* 1552) + 33 = 7795 words

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PSEC Data Frame - Offset values (decimal)** | | | |  |  |  |
|  |  |  |  |  |  |  |
|  |  | **PSEC0** | **PSEC1** | **PSEC2** | **PSEC3** | **PSEC4** |
| Startword | 0 |  |  |  |  |  |
| Frame ID | 1 |  |  |  |  |  |
| Preamble |  | 2 | 1554 | 3106 | 4658 | 6210 |
| ch0 data |  | 3 | 1555 | 3107 | 4659 | 6211 |
| ch1 data |  | 259 | 1811 | 3363 | 4915 | 6467 |
| ch2 data |  | 515 | 2067 | 3619 | 5171 | 6723 |
| ch3 data |  | 771 | 2323 | 3875 | 5427 | 6979 |
| ch4 data |  | 1027 | 2579 | 4131 | 5683 | 7235 |
| ch5 data |  | 1283 | 2835 | 4387 | 5939 | 7491 |
| Info0 |  | 1539 | 3091 | 4643 | 6195 | 7747 |
| Info1 |  | 1540 | 3092 | 4644 | 6196 | 7748 |
| Info2 |  | 1541 | 3093 | 4645 | 6197 | 7749 |
| Info3 |  | 1542 | 3094 | 4646 | 6198 | 7750 |
| Info4 |  | 1543 | 3095 | 4647 | 6199 | 7751 |
| Info5 |  | 1544 | 3096 | 4648 | 6200 | 7752 |
| Info6 |  | 1545 | 3097 | 4649 | 6201 | 7753 |
| Info7 |  | 1546 | 3098 | 4650 | 6202 | 7754 |
| Info8 |  | 1547 | 3099 | 4651 | 6203 | 7755 |
| Info9 |  | 1548 | 3100 | 4652 | 6204 | 7756 |
| Info10 |  | 1549 | 3101 | 4653 | 6205 | 7757 |
| Info11 |  | 1550 | 3102 | 4654 | 6206 | 7758 |
| Info12 |  | 1551 | 3103 | 4655 | 6207 | 7759 |
| Info13 |  | 1552 | 3104 | 4656 | 6208 | 7760 |
| PSEC end word |  | 1553 | 3105 | 4657 | 6209 | 7761 |
| Self trig rate ch0 |  | 7762 | 7768 | 7774 | 7780 | 7786 |
| Self trig rate ch1 |  | 7763 | 7769 | 7775 | 7781 | 7787 |
| Self trig rate ch2 |  | 7764 | 7770 | 7776 | 7782 | 7788 |
| Self trig rate ch3 |  | 7765 | 7771 | 7777 | 7783 | 7789 |
| Self trig rate ch4 |  | 7766 | 7772 | 7778 | 7784 | 7790 |
| Self trig rate ch5 |  | 7767 | 7773 | 7779 | 7785 | 7791 |
| Trig rate | 7792 |  |  |  |  |  |
| Frame ID | 7793 |  |  |  |  |  |
| Endword | 7794 |  |  |  |  |  |

**timestamp** = {6204, 4652, 3100, 1548}

**event count** = {3101, 1549}

# PSEC info words:

*Note: Some of the values are distributed across PSEC4 channels which makes it more difficult to reassemble the complete values.*

## Common parameters for all PSEC channels

Info 0: x”BA11”

Info 1: Wilkinson feedback count (current) - the DLL frequency measurement

Info 2: Wilkinson feedback target count setting – the required DLL frequency

Info 3: Vbias (pedestal) value setting

Info 4: Self trigger threshold value setting

Info 5: ‘PROVDD’ parameter setting

Info 11: VCDL count [15:0]

Info 12: VCDL count [31:16]

Info 13: ‘DLLVDD’ parameter setting

## Timestamps and counters

64-bit timestamp clocked at 320MHz – shows the time at which the trigger occurred.

Info 9, channel 0: timestamp [15:0]

Info 9, channel 1: timestamp [31:16]

Info 9, channel 2: timestamp [47:32]

Info 9, channel 3: timestamp [63:48]

Frame serial number – the total number of frames transmitted (of any type).

Info 10, channel 0: serial number [15:0]

Info 10, channel 1: serial number [31:16]

Event counter – the number of trigger events.

Info 10, channel 2: event count [15:0]

Info 10, channel 3: event count [31:16]

Beamgate counter – the number of beam gate trigger events.

Info 9, channel 4: beamgate count [15:0]

Info 10, channel 4: beamgate count [31:16]

## Trigger Info

This is useful as a confirmation that the board is setup as required.

Info 6 => trigger info 0

Info 7 => trigger info 1

Info 8 => trigger info 2

## trigInfo(Number, PSEC4 channel)

trigInfo(0,0) <= beamgate\_timestamp(63 downto 48);

trigInfo(0,1) <= beamgate\_timestamp(47 downto 32);

trigInfo(0,2) <= beamgate\_timestamp(31 downto 16);

trigInfo(0,3) <= beamgate\_timestamp(15 downto 0);

trigInfo(0,4)(15 downto 12) <= std\_logic\_vector(to\_unsigned(trigSetup.mode, 4));

trigInfo(0,4)(11 downto 10) <= trigSetup.sma\_invert & selfTrig.sign;

trigInfo(0,4)(9 downto 0) <= "00000" & std\_logic\_vector(to\_unsigned(selfTrig.coincidence\_min, 5));

--

trigInfo(1,0) <= "0000000000" & selfTrig.mask(0);

trigInfo(1,1) <= "0000000000" & selfTrig.mask(1);

trigInfo(1,2) <= "0000000000" & selfTrig.mask(2);

trigInfo(1,3) <= "0000000000" & selfTrig.mask(3);

trigInfo(1,4) <= "0000000000" & selfTrig.mask(4);

--

trigInfo(2,0) <= x"0" & std\_logic\_vector(to\_unsigned(selfTrig.threshold(0), 12));

trigInfo(2,1) <= x"0" & std\_logic\_vector(to\_unsigned(selfTrig.threshold(1), 12));

trigInfo(2,2) <= x"0" & std\_logic\_vector(to\_unsigned(selfTrig.threshold(2), 12));

trigInfo(2,3) <= x"0" & std\_logic\_vector(to\_unsigned(selfTrig.threshold(3), 12));

trigInfo(2,4) <= x"0" & std\_logic\_vector(to\_unsigned(selfTrig.threshold(4), 12));

## LED Standard Functions

### ACDC

|  |  |  |
| --- | --- | --- |
| 6 | 7 | 8 |
| 3 | 4 | 5 |
| 0 | 1 | 2 |

Red

Yellow

Green

Top: Serial Tx reset / selftrig mode beamgate trigger

Middle: Serial Rx PLL lock pps trigger

Bottom: Serial link ok FLL lock signal trigger

### ACC

|  |
| --- |
| 2 |
| 1 |
| 0 |

Red

Yellow

Green

Top: usb rx/tx

Middle: use external clock ref

Bottom: Serial link ok