Command Interface Specification for the ACC / ACDC Cards

Author: David Greenshields

Date: 14 May 2021

ACC V2.00

ACDC V2.00

# 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 standard UART protocol currently running at 10Mbaud.

Commands sent to the ACC over usb are 32-bit words. The 32-bit word is split into fields for acdc board select and command type, which are the top 8 bits and the next 4 bits of the word respectively. 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).

It should be noted that commands destined for the ACDC cards are first received by ACC’s usb port and then forwarded over the serial comms link to each ACDC board. Therefore after sending one command to acdc boards, a minimum delay time should be observed before sending another command. The commands are sent as 2 header bytes plus 4 instruction bytes so the transmission time will be around 6 microseconds. A suitable delay should be incorporated.

### Data Framing

Bytes sent from the ACDC card to the ACC, and from the ACC card to the control computer are combined in pairs to form 16-bit words.

Three types of data frames exist:

1. ACC: Local info data frame, 32 words

These are requested by the control computer. They contain information about acc setup, and in particular the number of data words received by each uart rx buffer.

1. ACDC: ID frame, 32 words

These are requested by the control computer. Used mainly as a method of detecting the presence of the ACDC card.

1. ACDC: PSEC4 data frame, 7795 words

These are generated automatically when a trigger event occurs and ‘transfer enable’ signal is high.

# 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

[15:12] = option 2

[11:0] = 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)

## 0x00030000 ACDC Board detect reset request

Resets the 8 ‘board detect’ bits to zero, indicating that no ACDC cards have been detected.

These bits are set whenever the uart rx buffer receives a valid frame from its respective ACDC board, i.e. one with the correct header and end of frame delimiter words.

## 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 Uart buffer read request

Causes the ACC to send the contents of the uart receive buffer of channel N (0 to 7) to the control computer.

## 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)

*Note: After switching to mode 9, the first pps trigger should be ignored as this could be a rising edge generated by switching to the pps signal which is already high. All subsequent triggers will be valid.*

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, if disabled no pps triggers will be generated but beamgate trigger fucntion

N = 0 disabled; No pps triggers, beam gate unction only

N = 1 enabled; pps triggers multiplexed with beam gate

## 0x004000XX LED mode select

The bits [5:0] select the mode for each led: 3 leds x 2 bits per led = 6 bits total

The 2-bit mode is as follows:

0 = standard led function

1 = led on

2 = led off

3 = test function (as selected by 0x0048XXXX command)

e.g.

0x00400000 = all leds are assigned their standard function, defined by firmware

0x00400055 = all leds on

0x004000AA = all leds off

0x004000FF = all leds in test mode

The front panel led indexes are:

2 top (red)

1 middle (yellow)

0 bottom (green)

## 0x0048NTXX LED test mode select

This sets the test function signal name when the led is in test mode.

N is the led number, 0 to 2

T is the monostable time: 8 = 500ms monostable, 0 = direct signal

XX is the signal number as follows:

ledMux(0) <= acdcBoardDetect(0);

ledMux(1) <= acdcBoardDetect(1);

ledMux(2) <= acdcBoardDetect(2);

ledMux(3) <= acdcBoardDetect(3);

ledMux(4) <= acdcBoardDetect(4);

ledMux(5) <= acdcBoardDetect(5);

ledMux(6) <= acdcBoardDetect(6);

ledMux(7) <= acdcBoardDetect(7);

ledMux(8) <= uartRx.buffer\_not\_empty(0);

ledMux(9) <= uartRx.buffer\_not\_empty(1);

ledMux(10) <= uartRx.buffer\_not\_empty(2);

ledMux(11) <= uartRx.buffer\_not\_empty(3);

ledMux(12) <= uartRx.buffer\_not\_empty(4);

ledMux(13) <= uartRx.buffer\_not\_empty(5);

ledMux(14) <= uartRx.buffer\_not\_empty(6);

ledMux(15) <= uartRx.buffer\_not\_empty(7);

ledMux(16) <= SMA(1);

ledMux(17) <= SMA(2);

ledMux(18) <= SMA(3);

ledMux(19) <= SMA(4);

ledMux(20) <= SMA(5);

ledMux(21) <= SMA(6);

ledMux(22) <= uartRx.valid(0);

ledMux(23) <= uartRx.valid(1);

ledMux(24) <= uartTx.enable(0);

ledMux(25) <= uartTx.enable(1);

ledMux(26) <= uartRx.bufferReset(0);

ledMux(27) <= uartRx.bufferReset(1);

ledMux(28) <= uartTx.valid;

ledMux(29) <= uartRx.error(0);

ledMux(30) <= uartRx.error(1);

ledMux(31) <= boardDetect\_resetReq;

ledMux(32) <= localInfo\_readReq;

ledMux(33) <= trig\_out(0);

ledMux(34) <= trig.sw;

ledMux(35) <= rxBuffer\_readReq;

ledMux(36) <= trig\_mux(0);

ledMux(37) <= LVDS\_in(0)(0);

ledMux(38) <= LVDS\_in(0)(1);

ledMux(39) <= LVDS\_in(0)(2);

ledMux(40) <= LVDS\_in(0)(3);

ledMux(41) <= usbRx.valid;

ledMux(42) <= usbTx.valid;

# 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.

This frame is ostensibly used for detecting the presence of the ACDC board. Note that PSEC4 frames should be disabled first using command 0xFFB54000 to prevent conflicts.

## 0xFFE0XXXX LED mode select

The bits [17:0] select the mode for each led: 9 leds x 2 bits per led = 18 bits total

The 2-bit mode is as follows:

0 = standard led function

1 = led on

2 = led off

3 = test function (as selected by 0xFFE8XXXX command)

e.g.

0xFFE00000 = all leds are assigned their standard function, defined by firmware

0xFFE55555 = all leds on

0xFFEAAAAA = all leds off

0xFFEFFFFF = all leds in test mode

The front panel led indexes are (from left to right):

6,7,8 top row (red)

3,4,5 middle row (yellow)

0,1,2 bottom row (green)

## 0xFFE8NTXX LED test mode select

This sets the test function signal name when the led is in test mode.

N is the led number, 0 to 8

T is the monostable time: 8 = 500ms monostable, 0 = direct signal

XX is the signal number as follows:

ledMux(0) <= '0';

ledMux(1) <= cmd.valid;

ledMux(2) <= FLL\_lock(0);

ledMux(3) <= transfer\_request;

ledMux(4) <= uartRx.valid;

ledMux(5) <= uartTx.valid;

ledMux(6) <= IDrequest;

ledMux(7) <= uartTx.dataTransferDone;

ledMux(8) <= digitize\_request;

ledMux(9) <= digitize\_done;

ledMux(10) <= phase\_mon;

ledMux(11) <= self\_trig;

ledMux(12) <= trig\_event;

ledMux(13) <= ‘0’;

ledMux(14) <= acc\_trig;

ledMux(15) <= sma\_trigIn;

ledMux(16) <= trig\_out;

ledMux(17) <= rampDone(0);

ledMux(18) <= trig\_clear;

ledMux(19) <= rampDone(0);

ledMux(20) <= dll\_resetRequest;

ledMux(21) <= transfer\_enable;

ledMux(22) <= PSEC4\_in(0).overflow;

ledMux(23) <= LVDS\_in(0);

ledMux(24) <= LVDS\_in(1);

ledMux(25) <= LVDS\_in(2);

ledMux(26) <= trig detect;

ledMux(27) <= trig valid;

ledMux(28) <= trig abort;

ledMux(29) <= pps detect;

## 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.

To detect the presence of ACDC boards, a short (32 word) frame is requested from all boards. If this frame is received by the uart buffer, the corresponding ‘board detect’ bit will go high automatically. The received frame may be read but it is not necessary. The board detect bits are available in the acc local data frame.

***Procedure:***

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 0x00030000 // reset all of the board detect bits to zero

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

Each ACDC card can have its own trigger mode.

Trigger mode setting procedure example:

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.

For more complex setups, the following example shows how to write different modes to different ACDC cards:

Send 0x003000F1 //ACC trig setup: Set trigger mode 1 for ACDC boards 0 to 3

Send 0x0FB00001 //ACDC trig setup: Set trigger mode 1 for ACDC boards 0 to 3

Send 0x00300F05 //ACC trig setup: Set trigger mode 5 for ACDC boards 4 to 7

Send 0xF0B00005 //ACDC trig setup: Set trigger mode 5 for ACDC boards 4 to 7

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 0xFFB180NN //set up self trigger use coincidence

### ACDC SMA

send 0xFFB2000N //set up ACDC SMA invert

## 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 uart 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 rx buffer length (words received) for each channel
   3. If rx buffer length is equal to psec frame length (7795) then record the data
4. Record the data:
   1. Request a read of ACDC data on the corresponding channel
   2. Read the data frame from the ACC uart buffer of the same channel
   3. When all channels done, loop back to step 2

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

# ACC Local Info 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) <= acdcBoardDetect [7:0]

localData(8) <= pps mux enable [1]; SMA invert [0]

localData(9) <= x”0000”;

localData(10) <= PLL lock fail counter [15:0]

localData(11) <= x"0000";

localData(12) <= PLL lock [1]; useExtRef [0]; -- bit 0 goes high if ext ref is selected

localData(13) <= x"0000";

localData(14) <= x"0000";

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) <= x"0000";

localData(25) <= x"0000";

localData(26) <= x"0000";

localData(27) <= x"0000";

localData(28) <= x"0000";

localData(29) <= x"0000";

localData(30) <= x"AAAA";

localData(31) <= x"4321";

*Notes:*

Word 0: frame start delimiter

Word 1: frame type ID

Word 2,3,4: version info

Word 7: 8 bit field containing board detect info (1=detected, 0= not detected) for each ACDC card

Word 12: ext ref detect. This will be fixed at 1 or 0 after reset and will not change until power cycle

Word 16-23: Shows the frame length (number of words received) for each uart rx buffer

Word 30: frame type ID

Word 31: end of frame delimiter

# 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"0000";

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) <= pps trigger count [31:16]

IDframe\_data(12) <= pps trigger count [15:0]

IDframe\_data(13) <= beam gate count [31:16]

IDframe\_data(14) <= beam gate count [15:0]

IDframe\_data(15) <= x"0000";

IDframe\_data(16) <= x"0000";

IDframe\_data(17) <= x"0000";

IDframe\_data(18) <= x"0000";

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) <= x"0000";

IDframe\_data(29) <= x"0000";

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

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

*Notes:*

Word 0: frame start delimiter

Word 1: frame type ID

Word 2,3,4: version info

Word 9: wilkinson feedback current count value for channel 0 (for info only)

Word 10: wilkinson feedback target count value for channel 0 (for info only)

Word 30: frame type ID

Word 31: end of frame delimiter

# PSEC Data Frame Structure

## Once per frame:

0x1234 Startword

0xA5EC Frame type ID (Note: for PPS frames this changes to 0xABCD)

## 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

## Timestamp and event count

Info 9: 64-bit timestamp clocked at 320MHz – shows the time at which the trigger occurred.

PSEC0: timestamp [15:0]

PSEC1: timestamp [31:16]

PSEC2: timestamp [47:32]

PSEC3: timestamp [63:48]

Info 10: Event counter – counts the number of trigger events.

PSEC0: event count [15:0]

PSEC1: event 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

see below:-

## trigInfo(Number, PSEC4 channel)

trigInfo(0,0) <= x"EEEE";

trigInfo(0,1)(3 downto 0) <= std\_logic\_vector(to\_unsigned(trigSetup.mode, 4));

trigInfo(0,2) <= 0

trigInfo(0,3)(1 downto 0) <= trigSetup.sma\_inver[1]

trigInfo(0,3)(5 downto 4) <= trigSetup.selfTrig\_sign[5]

trigInfo(0,3)(15 downto 6) <= "00000" & std\_logic\_vector(to\_unsigned(trigSetup.selfTrig\_coincidence\_min, 5));

trigInfo(0,4) <= x"EEEE";

--

trigInfo(1,0) <= "0000000000" & trigSetup.selfTrig\_mask(0);

trigInfo(1,1) <= "0000000000" & trigSetup.selfTrig\_mask(1);

trigInfo(1,2) <= "0000000000" & trigSetup.selfTrig\_mask(2);

trigInfo(1,3) <= "0000000000" & trigSetup.selfTrig\_mask(3);

trigInfo(1,4) <= "0000000000" & trigSetup.selfTrig\_mask(4);

--

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

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

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

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

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

## LED Standard Functions

### ACDC

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

Red 8 Firmware PLL lock and burst flash (1Hz & 10Hz clock)

Red 7 DLL reset or Trigger reset

Red 6 Serial Tx

Yellow 5 Self-trigger mode

Yellow 4 Jitter cleaner PLL locked

Yellow 3 Serial Rx

Green 2 DLL lock and burst flash (1Hz & 10Hz clock)

Green 1 Trigger event

Green 0 Command received

### ACC

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

Red 2 usb Tx

Yellow 1 usb Rx

Green 0 ACDC board detect (channel 0)

# Version history

## 14/5/2021

Added PPS / beamgate multiplex signal.

## ACC v2.00

Frame words changed – see word 8,9 10 & 12 above.

Command handler: trigger cmd 0031xxxx to 0035xxxx changed.

Level detection no longer used.

## ACDC v2.00

32 word frame: added words 11-14 for pps trig count and beam gate count

Long data frame: now can be a PSEC or PPS data frame with dummy PSEC data

Indicated by the value in word 1. (see above)

Command no longer used: B1700x, B3000x, B4000x, B5300x

New led test signals available for trigger mechanism

Level detection no longer used (edge detect only)

## 12 Feb 2021

1. Command 0x003200xx no longer available to set the trigger source for each acdc simultaneously, use command 0x00300xxx instead.
2. New mode added: trigger mode 9. This sets the trigger source to the pps signal coming from the system lvds connector.
3. The trigger source bits in the ACC 32 word frame in word 9 are now 2 bits per channel instead of 1, so ch 7 is bits [15:14] .... ch 6 is bits[13:12] etc. The 2-bit mode is:

00 software trigger

01 hardware trigger (sma)

10 hardware trigger (pps)

11 null