Version 1
August 15, 2017

Documentation of Firmware Functionality

Horatio Li
The University of Chicago

Abstract

This documentation is about how the software commands are processed in the firmware level of ACC and ACDC boards. The ACC is interfaced to the computer via USB, which sends command words (C++ USB instructions) to the USB module inside ACC for instructions. These instructions are designed to serve such functions as sending triggers, aligning ACC and ACDC boards, resetting various signals, and transmitting data between boards.

In the ACC firmware, this is done by assigning different hexadecimal values to a signal internal to the USB module called USB_INSTRUCTION (specifically, USB_INSTRUCTION is a signal consisting of 32 bits, and the hex values are assigned to the 16th through 19th bits of the 32 bit-field). Essentially, this signal carries information from the C++ command word and then, with different hex values, triggers a set of different signals to send out information to other modules in the ACC or to the ACDC. In total, there are five hexadecimal values of the 16th to 19th bits of USB_INSTRUCTION that correspond to specific functionalities.

Hexadecimal Values	Corresponding Binary	Corresponding Functionalities
E	1110	Software Trigger
D	1101	Align LVDS
C	1100	CC Read Mode
B	1011	Prepare Sync
4	100	Hardware Reset

The following sections are explanations for how each of the above functionalities work in the firmware level. Each of them contains a signal chart indicating the following information about the relevant signals 1) location (where the signals are involved in the firmware, this usually consists in module name + line number); 2) description and functionality (which includes firstly a detailed trace of where the signal goes and comes from, and secondly a summary of the general role the signal functions in the corresponding functionality).

1 Software Trigger

The following is the bit-field range of USB_INSTRUCTION for this functionality:

Bit Range	Name	Description
19-16	**0xE**	Command marker
6-4	SOFT_TRIG_BIN	
3-0	SOFT_TRIG_MASK	

The following is the ”destination” part of the firmware:

```
when x"E" =>
        SYNC_TRIG <= '1';
        SOFT_TRIG <= '1';
        SOFT_TRIG_MASK(3 downto 0) <= USB_INSTRUCTION(3
           ↪ downto 0);
        SOFT_TRIG_MASK   <= (others=>'1');  --trigger
           ↪ every AC/DC
        SOFT_TRIG_BIN    <= USB_INSTRUCTION(6 downto 4);
             if delay > 8 then
                  delay := 0;
                  state <= st1_WAIT;
             else
                  delay := delay + 1;
             end if;
```

1.1 Introduction

This functionality starts with a synchronization process in the USB module, which is enabled by turning a signal called cc_only_instruct_rdy high. The sync process essentially prepares the software trigger related signals (SOFT_TRIG_MASK, SOFT_TRIG_BIN) to send out to other modules.

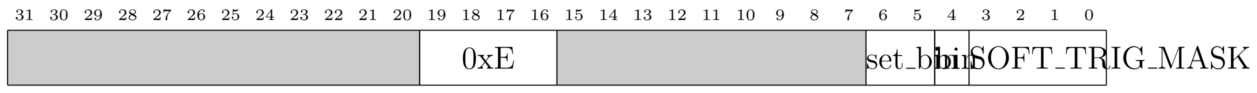

Figure 1: Command 0xE bit fields

1.2 In TriggerAndTime Module

The only two output signals from the USB module are SOFT_TRIG_MASK and SOFT_TRIG_BIN. The first serves as a mask which determines which ACDC board(s) to send dedicated trigger line from the ACC_main module (which will be discussed later in this documentation); the second functions to enable SOFT_TRIG_MASK to be sent to ACC_main and to enable a system clock counting process.

2 CC Read Mode

2.1 Introduction and bit-field range

The CC read mode functionality, initialized in the USB interface by software command words, sends instructions / signals for two major uses: 1) instructions directly related to read cc mode (CC_INSTRUCTION) gets sent to the ACCmain module and then eventually to various places in ACDC boards; 2) signals related to triggers that get sent to triggerAndTime and through the dedicated trigger line (xDCout_(1)) to ACDC boards. In the following documentation, the first section describes the former while the second section describes the latter.

The following is the bit-field range of USB_INSTRUCTION in this functionality:

Bit Range	Name	Description
19-16	**0xC**	Command marker
14-12	SET_TRIG _DELAY	
11-5	TRIG_DELAY	
3	TRIG_MODE	
2-0	CC_READ_MODE	

2.2 Initialization: Synching CC instruction signals inside USBWrapper module

The CC read mode functionality is processed when we enter (at the software level) 0xC (binary 1100) to the 19-16 bits of signal USB_INSTRUCTION. In the USBWrapper module (where the software command enters the ACC), we initialize this functionality by

1) setting a signal called CC_INSTRUCT_RDY high and

2) mapping the value of USB_INSTRUCTION to CC_INSTRUCTION (see line 815 819 USBWrapperACC).

The first result enables two signals to be synchronized and thus prepared to be sent to other modules for further purposes related to cc read mode functionality. The result of this synchronization process is the following: 1) CC_INSTRUCTION is sent to a port called xInstruction [31..0] in ACCmain module; 2) INSTRUCT_MASK (which determines which ACDC board(s) to send instruction) is sent to a port called xfe_mask[7..0] in ACCmain module; 3) a signal called instruction_rdy is turned on, which sets xInstruct_Rdy in ACCmain module high.

The second result basically enables software commands (USB_INSTRUCTION) to set values to all the digits of CC_INSTRUCTION.

2.3 Signals that have gone into ACC_main from usbWrapperACC

- xInstruct_ready
- xInstruction[31..0]
- xfe_mask[7..0]

2.3.1 Functionality of xInstruct_ready

Instruct_Ry is mapped to a signal called xCC_INSTRUCT_RDY which functions to enable instruction to be ready to send to the desired front-end board. In the component transceivers, we are interested in sending instruction to the i-th ACDC board by masking via xDC_MASK which is mapped to the i-th digit of xfe_mask.

In the transceivers component, this is done by process (line 258) in which 32 bit word are sent 8 bits at a time. It firstly sends STARTWORD_8a and STARTWORD_8b via GOOD_DATA. Then it sends xCC_INSTRUCTION (which are CC-related instruction that is sent from the USBwrapper) to DCout_(0).

2.3.2 Functionality of xInstruction[31..0]

xInstruction is sent to xCC_instruction in a component called transceivers (mentioned above). In that component, as described in the above section, it gets sent 8 bit at a time by a process in line 258 (in which a signal called GOOD_DATA serves as an intermediate link that carries xCC_instruction to TX_DATA in a sub-component called lvds_transceiver, in which). Eventually these signals go to lvds_com as xRX_LVDS_DATA.

2.3.3 Functionality of xfe_mask[7..0]

xfe_mask goes to xDC_mask in the TRANSCEIVER component. It shows up in the process mentioned in the descriptions of the two signals above, in which it functions to enable the data-processing to the selected desired front-end boards.

2.3.4 Summary of 2.3

In general, the first signal xInstruct_Rdy serves as a preliminary condition under which the second signal, xInstruction, can be sent to the desired ACDC boards selected by the third signal xfe_mask.

Note This part of the functionality CC Read Mode is similar to software trigger in that it also involves a process of selecting the desired front-end ACDC boards and then send data (instructions) to it. What distinguishes this functionality from the other is that this functionality processed xDCout_(0) while the software trigger functionality processed xDCout_[2..1]. While the 2 and 1 digit of xDCout take care of aligning the ACC with ACDC AND triggering ACDC from ACC, digit 0 of xDCout sends out lvds data received by the ACDC.

2.4 In ACDC: Instructions from ACC via LVDS

General Description The signal xDCout_[0] is mapped to a port on module lvds_com called xRX_LVDS_DATA, which is linked eventually to an internal signal called RX_DATA, whose functionality is shown in 2 processes: the first one starting on line 181 and the second starting on line 246 (both in module lvds_com).

2.4.1 Process 181: Alignment & Readout

In this process, the preliminary requirement involves xALIGN_ACTIVE=1, which we have from align_lvds functionality (NOTE: in align_lvds functionality, signal xtrig is mapped to xDCout_(1), which goes to xalign_active in ACDC lvds_com). The purpose of this process is to align RX_DATA with a signal called ALIGN_WORD_8, which is a 8-bit signal whose concatenation with itself is assigned to TX_DATA (i.e. TX_DATA = ALIGN_WORD_8 & ALIGN_WORD_8). In other words, this process makes sure that RX_DATA is aligned with some external threshold or standard (ALIGN_WORD). After this process, TX_DATA (which is the concatenation of RX_DATA with itself) is transmitted to an output port called xTX_LVDS_DATA[1..0] which is sent to the ACC_main in xDCin_[1..0] for readout.

After sent back to ACC_main: (xDCin_[1..0]) The signal xDCin_[1..0] is mapped to a port in ACC_main called xRX_LVDS_DATA, which is eventually mapped to an internal signal called RX_DATA in the transceivers component within ACC_main.

Inside the transceivers component, RX_DATA[15..0] is divided into two parts (which essentially correspond to rx_serdes(1) and rx_serdes(0), each undergoing a check for alignment process (note: this check for alignment process occurs previously with TX_DATA as well: see process 181 above). Then, RX_DATA as a whole is mapped to an internal signal called CHECK_RX_DATA. CHECK_RX_DATA enables either START_WRITE or STOP_WRITE (which will be explained in the following). This means that the data we receive from the ACDC boards (i.e. RX_DATA) enables us to store it into RAM and eventually for USB readout:

If START_WRITE is high (which means we are in write mode), we store RX_DATA to RX_DATA_TO_RAM. In this case, we store RX_DATA to a signal called xRxData in ACCmain, then is sent to a signal called xADC[7..0][15..0] in the USB. Then, in the USB it gets selected by FPGA_DATA (for specific procedure: see packetUSB.vhd line178) and then sent out by a external port in USBwrapper called FD to the computer.

2.4.2 Process 246: passing instructions to decode_instruct module

When we finish the previous process, we have ALIGN_DONE as the state of the process, under which ALIGN_SUCCESS is high. Given ALIGN_SUCCESS = 1, if RX_DATA is aligned with a threshold (similar to the "check for alignment" process in 181, as shown above), we assign RX_DATA to CC_INSTRUCTION by concatenating RX_DATA 4 times. The result of this process, eventually, is that: 1.we set the four times concatenation of RX_DATA to CC_INSTRUCTION, which is again mapped to a signal called CC_INSTRUCTION_FULL (this eventually goes to **xINSTRUCT_WORD[31..0]** in decode_instruct); 2. we set INSTRUCT_READY high (which turns xINSTRUCT_READY high, and this goes to xINSTRUCT_FLAG in module decode_instruct)

2.4.3 Various Instructions given by xINSTRUCT_WORD in decode_instruct

NOTE: this process gives out 11 possible instructions to ACDC xINSTRUCT_WORD goes to process 238 in decode_instruct which is initiated by xALIGN_SUCCESS = 1, a condition satisfied by process 181 above. Then, a second initiation is set by xINSTRUCT_FLAG (which comes from process 246 above) going high. This second initiation then allows the assignment of xINSTRUCT_WORD to various other internal signals. The process then starts with different cases of INSTRUCTION:

Instructions	Signal Used	Output Signals
set_dll_vdd	INSTRUCT_PSEC_MASK(j)	when high, SET_DLL_VDD(j) (15..0) ¡= 0x0&INSTRUCT
2	7	78
3	545	778
4	545	18744
5	88	788

2.5 Signals that have gone into triggerAndTime from usbWrapperACC

Given the condition that bit 4 of USB_INSTRUCTION is high, we assign values to the following signals by mapping them to corresponding digits in USB_INSTRUCTION: TRIG_MODE, TRIG_DELY(6..0), SET_TRIG_SOURCE(2..0). Respectively, they get mapped to the following signals in triggerAndTime module:

- **xMODE**: it becomes the ninth bit in the array xBIN_COUNT. It functions similarly to a reset signal: if xMODE is zero, some trigger-related signals like LATCHED_TRIG goes to zero.
- **xTRIG_DELAY**: Only appears in line 275 which is commented out. This therefore seems useless.
- **xTRIG_SOURCE[2..0]**: INTERNAL_TRIGGER (an internal signal of triggerAndTime module, which is discussed below) can be set to several different things based on xTRIG_SOURCE.

2.5.1 Functionality of these signals

Setting LATCHED_TRIG high: xMODE and xTRIG_SOURCE converge at a process in line 219 in which an internal signal called LATCED_TRIG is set high. The two signals we are interested in are involved in the following way: if 1) xMODE is set high, 2) xTRIG_SOURCE(0) is low (so that a signal called USE_BEAM_ON is low: see line 208), and 3) an external signal called xEXT_TRIG_VALID is high (explained below), then LATCHED_TRIG is high.

(NOTE: xEXT_TRIG_VALID is set high by a signal called xtrig_valid in usbWrapperACC. xtrig_valid should be set high by trig_valid_cc_only, which is set high by the cc read mode functionality. Yet Oberla seems to comment this out so we are not sure how xEXT_TRIG_VALID can be set high)

What LATCHED_TRIG does: The specific functionality of LATCHED_TRIG seems unclear now. In triggerAndTime module, LATCHED_TRIG turned high results in the following:

- counting of AUX_TRIG_1_counter and AUX_TRIG_0_counter

- mapping of AUX_TRIG_2 and AUX_TRIG_3 to some output ports
- CLOCKED_TRIG goes high

Yet all of the signals/ports involving AUX_TRIG dont go anywhere (their output ports are cut off in Oberlas ACC_top_v0_1.bdf file). The only useful outcome is CLOCKED_TRIG, which enables xTRIG_OUT to go high for all 8 ACDC boards.

2.5.2 Summary of 2.5

The only outputting signal from triggerAndTime module, related to functionality cc read mode, is xTRIG_OUT. This signal goes to a port called xtrig[7..0] in ACC_main, which gets mapped to an output signal called xDCout_(1), which goes to lvds_rx_in[1] in decode_instruct module of ACDC boards. There, it functions essentially as a global reset signal (see line 131 132 in decode_instruct).

3 Align LVDS

3.1 Introduction and bit-field range

This functionality is designed to align ACC and ACDC boards. This is done in two aspects:

- It aligns the signals that get sent to / get sent by the ACDC boards from / to the ACC board
- It aligns the two boards such that a global reset signal can be passed from the ACC to the ACDC board to trigger a clear-all process in the ACDC

The signal ALIGN_LVDS_FLAG is turned on by the USB command D. After being sent via the SERDES line (ALIGN_LVDS_FLAG is mapped to xDCout(2) in ACC_main module) to the ACDC, it gets mapped to two different places: module **decode_instruct** and **lvds_com**.

The following is the bit-field range of USB_INSTRUCTION for align lvds functionality:

Bit Range	Name	Description
19-16	**0xD**	Command marker

3.2 In decode_instruct: Enabling Global-Reset ACDC

The signal is mapped to an external port of this module called xALIGN_ACTIVE, which serves essentially as a flag which indicates that the alignment is good between the two boards. With this signal high, a hardware reset signal from the USB of ACC (this is how the functionality Hardware reset comes into play: it sends a signal called xTRIG_FROM_SYS from the ACC board to give a global reset command to the ACDC board) is able to turn the global reset signal high, resulting in a global clear-all signals occurring in all the different modules within the ACDC.

3.3 In lvds_com: Enabling data-transmission

The signal is mapped to an external port of this module called xALIGN_ACTIVE, which functions (in line 177 specifically) to enable the received data from ACC (RX_DATA) to be checked (the specific criteria this is checking can be seen in the documentation on cc_read_mode functionality, here it is involved only because this checking process is enabled by align_lvds functionality). Eventually, this enables data to be transmitted back to ACC for readout.

Another function that align lvds serves in this module can be seen in line 249, in which it enables RX_DATA (instruction data from ACC) to map to CC_INSTRUCTION, which eventually goes to a port in decode_instruct called xINSTRUCT_WORD. Significantly, this signal gives various instructions to the ACDC board depending on the different CC_INSTRUCTION it receives. For specific instructions, see documentation on cc_read_mode.

4 Hardware Reset

4.1 Introduction and bit-field range

This functionality is intended to set a global reset signal in both the ACC and the ACDC boards. This is achieved by signals called HARD_RESET and RESET_DLL_FLAG which serve as such global reset signals in the ACC and by applying certain signals (SOFT_TRIG, ALIGN_LVDS_FLAG, etc.) related to the software trigger and align lvds functionalities.

The following is the bit-field range of USB_INSTRUCTION for Hardware Reset functionality:

Bit Range	Name	Description
19-16	**0x4**	Command marker
11-0	0xFFF, 0xEFF	
15-12	0x1, 0x3, 0xF	Controls RESET_DLL_FLAG

4.2 11-0: global reset

In general, this part functions as a hardware reset which resets processes/signals within the ACC board. It further functions as a hardware reset of the ACDC board (this is done by using some functionality involving software trigger and align lvds).

- Software trigger: When USB_INSTRUCTION(15..12) is FFF (twelve binary digits all set to 1), we set three signals related to software trigger to high: SOFT_TRIG, SOFT_TRIG_MASK, and cc_only_instruct_rdy. This enables a dedicated trigger line (xDCout_(1)) to be sent to all the (8) ACDC boards. In each ACDC board, this signal functions (together with xDCout_(2) set high)as a global reset.
- Align_lvds: to satisfy the condition (xDCout_(2) set high), ALIGN_LVDS_FLAG is set high. Together, they turns global reset on, which (in the clock main module of ACDC) turns on a clear-all signal that resets a lot of processes/signals in all the modules within the ACDC.

- hard_reset: When USB_INSTRUCTION(15..12) is FFF (twelve binary digits all set to 1), a signal called HARD_RESET is set high, which serves as a global reset/clear-all signal that goes to USBwrapper and triggerAndtime. **NOTE: for what specific tasks the global reset signal performs, see Software-trigger functionality**

Note: the clear-all design in both the ACDC (software trigger and Align_lvds) and the ACC (HARD_RESET) involves a similar process of counting clock signals. The specific processes, respectively, can be seen in line 72 115 (progreset component in CLKain in ACDC) and line 119 157 (progreset component in ACC_main). In both processes, a signal (PULSE_RES in the ACC for example) is set high (in this case by HARD_RESET). When this happens (which is its rising_edge), we start counting 100000 clock signals (in both situations, CLK). In this counting process, an intermediate signal (in the ACC, xRESET_SIG(2)) is set high until the counting stops. This intermediate signal in turn triggers the global-reset signal.

4.3 15-12: Reset_DLL_Flag

In general, there are three circumstances under which an internal signal called RESEST_DLL_FLAG is set high: when the 15 downto 12 digits of the bit field is 0001, 0011, or 1111.

RESET_DLL_FLAG is sent out to a port called xEVENT_AND_TIME_RESET in the triggerAndTime module. In this module, it functions (when set high) as a clear-all signal that sets a bunch of signals to zero. *NOTE: It is unclear what further significance RESET_DLL_FLAG has, since the signals that are set zero are not closely related to each other or to a specific use (see Signal Charts for reference).*

5 Prepare Sync (This needs revision: don't really understand this)

5.1 Introduction and bit-field range

This functionality involves initializing other two functionalities (software trigger and cc read mode) by preparing for them a synchronization of related signals (such as soft_trig for software trigger functionality).

The following is the USB_INSTRUCTION bit-field range for this functionality:

Bit Range	Name	Description
19-16	**0xB**	Command marker
1	trig_valid	If USB_INSTRUCTION(2) = 1
3	CC_SYNC	If USB_INSTRUCTION(4) = 1
0	CC_SOFT_DONE	If USB_INSTRUCTION(4, 2) = 0

5.2 Three cases

In line 834 of the usbWrapperACC module, the process starts with different cases of USB_INSTRUCTION, and results in different outcomes of 1. cc_only_instruct ready and 2. cc instruct ready and 3. the value of trig_valid flag (which gets sent to triggerAndtime module).

5.2.1 Enabling Synchronizations?

Signal Description: When bit 2 of USB_INSTRUCTION goes high, the bit-field is assigned as the follows: the value of bit 1 of USB_INSTRUCTION is assigned to trig_valid (which is outputted to trigger and time module), and the value of USB_INSTRUCTION is assigned to CC_INSTRUCTION. In addition, we have two outcomes: ready_for_cc_instruct and ready_for_instruct are set high. (NOTE: in general, this part has the following primary outcomes: the value of cc_only_instruct_rdy and cc_instruct_rdy; the value of cc_instruction (whether it goes to 0s or has the value of USB_INSTRUCTION); the value of trig_valid)

Summary: In the context of the usbWrapperACC, the first signal (ready_for_cc_instruct) set high enables the functionality software trigger (when 0xE) to run. In line 536, only when ready_for_cc_instruct = 1 can the SOFT_TRIG signals get synchronized. Similarly, the second signal (ready_for_instruct) set high enables the functionality cc read mode (when 0xC) to run. In line 635, only when ready_for_instruct = 1 can the CC_instruction signals get synchronized.

In other modules (TRIG_VALID, SYNC_MODE): trig_valid goes to the port xEXT_TRIG_VALID in triggerAndtime module. In general, it functions as a flag which enables an internal signal called LATCHED_TRIG to be turned on. In the general context of the DAQ functionalities, this enables one branch of the functionality cc_read_mode to run (part of cc_read_mode functionality (see line 225 triggerAndTime) relies on sending a signal called xTRIG_OUT out of the triggerAndTime module into the ACC_main module, for specific reference see documentation on cc_read_mode).

As an internal signal, SYNC_MODE doesnt get mapped anywhere. This seems to be useless.

5.2.2 Stop Synchronization

Signal Description: When bit 4 of USB_INSTRUCTION is high (and bit 2 is low), both cc_only_instruct_rdy and cc_instruct_rdy are set low. In addition, CC_INSTRUCTION is set to 0s. This therefore seems to be a reset/clear-all type of function.

Summary: This disenables both software trigger and cc read mode functionalities.

In other modules (CC_SYNC): This signal is mapped to a signal called CC_SYNC_REG. This signal functions as a flag in the synchronization processes for software trigger and cc read mode. When turned high, this indicates that the signals are not synced yet and have to go through synchronizations. When turned low, it indicates that the signals are ready to be sent out.

5.2.3 ???

Signal Description: he remaining case is that if USB_INSTRUCTION(4) and (2) are both low, then we disenable software trigger functionality and enable cc_read_mode functionality (in the firmware level, this is done by setting CC_INSTRUCTION ¡= USB_INSTRUCTION (which enables USB_INSTRUCTION to be sent via xDCout_(0) to the front-end boards) and by setting CC_INSTRUCT_RDY ¡= 1).

NOTE: The unique part of this process is that CC_SOFT_DONE is mapped to USB_INSTRUCTION(0). In a larger scale, this signal is sent to a port called xready in the ACC_main module.

In other modules(CC_SOFT_DONE): In the ACC_main module, this signal gets sent to an internal signal (inside the component transceivers) called xSOFT_RESET. When this signal goes high (which means USB_INSTRUCTION(0) is high) it disenables the cc_read_mode functionality to get data from the ACDC boards (refer to line 313 and 355 in transceivers.vhd). Of course, when USB_INSTRUCTION(0) is low, this enables the cc_read_mode functionality to run. In general, it serves as a software reset which indicates that the software is done reading to cpu and therefore either sets all the read-data relevant data to low or blocks the cc_read_mode functionality (as described above).

References

[1] http://psec.uchicago.edu/library/data/Margherita/M12/photos/