LAC Reference Manual

March 25, 2009

Release 1, version 0 (R1V0)

Gabriele Bigongiari

1 Data Format

The LAC board is connected to a PC with a QuickUsb module via th USB-2 protocol. This module can transfer a block of data values from the board RAM to PC, using the QuickUsbReadData command (see [1] at page 57). The LAC is configured to transfer a data block of 512*9 (= 4608) bytes. This configuration allows the LAC to read up to 16 VA boards (VAB). Each VAB has 128 channels with 16 digits ADC data.

The data block can contain an event consisting of a header, up to 16 VAB frames and an event trailer (see figure 1). The event header is 8 bits wide and it consists of three fields:

1. EVENT TAG: 16 bits constant value 0x5A5A, marking the starting point of an event;
2. EVENT NUMBER: 32 bits number indicating the number of the event;
3. TRIG TYPE: 16 bits fields; only 4 LS bits are used; each bit corresponds to a different trigger type.

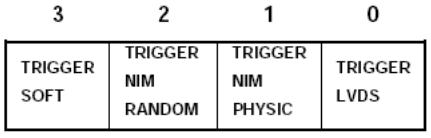

Figure 2: trig type field

After the event header there are 16 VAB frames. Each VAB frame consists of 130 fields:

1. VAB TAG: 16 bits constant value 0xF0F0, marking the starting point of a VAB frame;
2. VAB ID: 16 bits field; only the four LS bits are used; up to 16 VAB boards can be connected to one LAC;

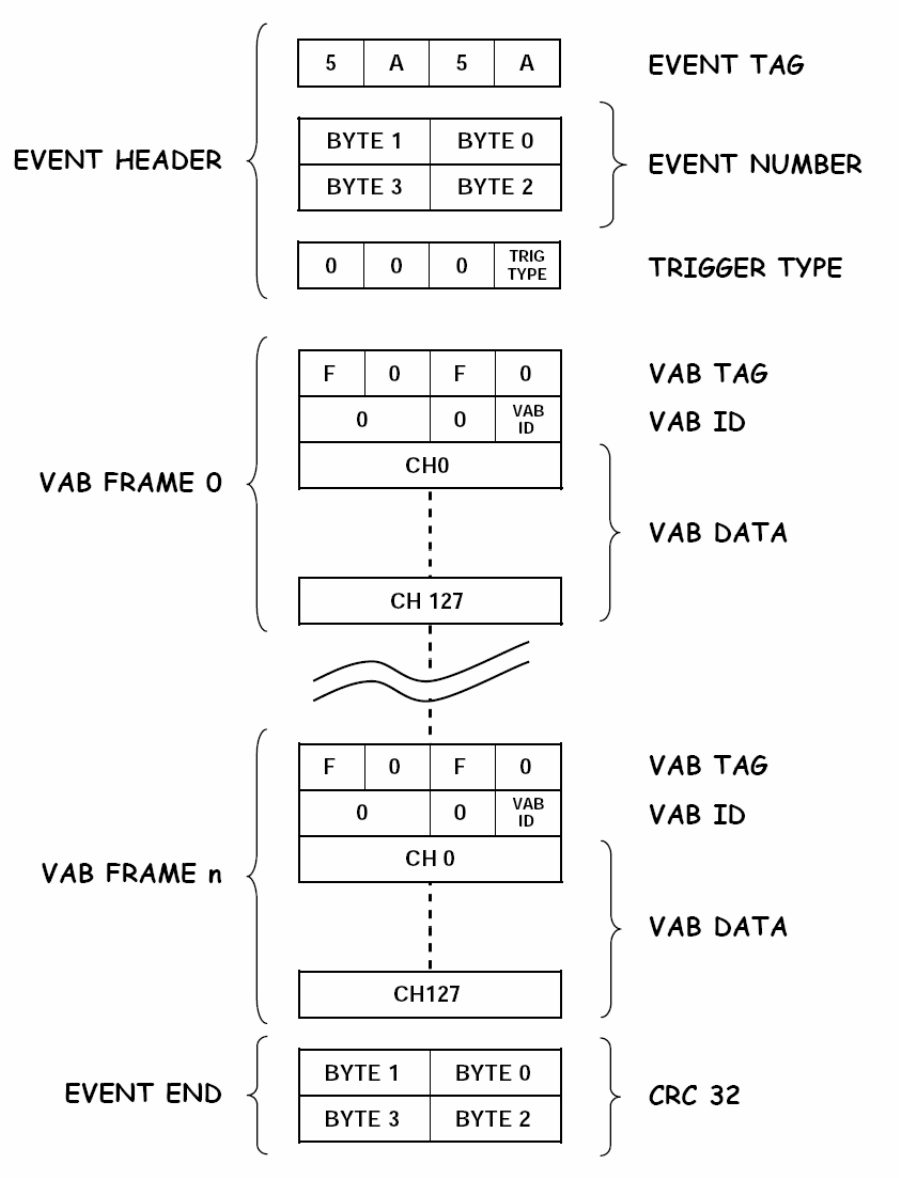

Figure 1: Data Format scheme

3. CH0 -> CH128 fields: 16 bits numbers, each corresponding to a VA board channel.

At the end of the event there is a 32 bits fields (CRC32), containing the 32 bits Cyclic Redundancy Check of the whole event (at present the check is not implemented and the field is set to the constant value 0x0088). The total length of a LAC event is calculated by adding the event header length (8 bytes), the VAB frames length (16*(128*2 + 4) = 4160 bytes) and the event trailer length (4 bytes), that is 4172 bytes. Note that a VAB frame, corresponding to a void LAC connector, will have all the channels set at zero value.

2 Registers Map

The QuickUSB library (see [1]) permits to send commands to the LAC and read status of a few parameters of the card by writing/reading some registers. The

Address	Name	R/W	Note
0x0000	BOARD ID	R	
0x0001	HIGH VOLTAGE VALUE	R/W	10 bits wide
0x0002	HIGH VOLTAGE RAMP UP	R/W	not implemented
0x0003	HIGH VOLTAGE RAMP DOWN	R/W	not implemented
0x0004	TRIGGER SOFT	R/W	
0x0005	TRIGGER STATUS	R	
0x0006	TRIGGER CONF	R/W	
0x0007	LAC CONF	R/W	
0x0008	LADDER MAP	R	
0x0009	HOLD VALUE SET	W	

Table 1: register table

list of registers is resumed in table 1. There are 10 read/write registers:

1. BOARD ID - ADDRESS 0x0000

   this is an only read register with a 16 bits range;

   it returns the LAC ID.

2. HIGH VOLTAGE - ADDRESS 0x0001

   this is a 10 bits range register;

   the value wrote in this register is sent to serial DAC that sets the high voltage generator.

3. HIGH VOLTAGE RAMP UP - ADDRESS 0x0002

   not implemented yet.

4. HIGH VOLTAGE RAMP DOWN - ADDRESS 0x0003

   not implemented yet.

5. TRIGGER SOFT - ADDRESS 0x0004

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TS

Figure 3: trigger soft register

this is a 16 bit range write only register (see the configuration of the bits in fig. 3);

when it is read it returns always the value 0x0000;

If we write the value "1" in the TS position (bit number 0) we can force a SOFTWARE TRIGGER (if bit 3 in the TRIGGER CONF register is up).

6. TRIGGER STATUS - ADDRESS 0x0005

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	EVENT READY	0	TRIG SOFT	TRIG NIM_R	TRIG NIM_P	TRIG LVDS

Figure 4: trigger status register

this is a 16 bit read only register;

in fig. 4 the bits configuration is shown; there are 5 trigger flag bits;

each flag bit set to 1 means respectively:

- TRIG LVDS (bit 0) = 1 → there is a trigger on LVDS trigger input of the LAC;
- TRIG NIM_P (bit 1) = 1 → there is a trigger on NIM physical trigger input of the LAC;
- TRIG NIM_R (bit 2) = 1 → there is a trigger on NIM random trigger input of the LAC;
- TRIG SOFT (bit 3) = 1 → there is a software trigger;
- EVENT READY (bit 5) = 1 → the event is ready to be transferred from LAC.

7. TRIGGER CONF - ADDRESS 0x0006

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRIG CLR	0	0	0	0	0	0	0	0	POL NIM_R	POL NIM_P	POL LVDS	MASK SOFT	MASK NIM_R	MASK NIM_P	MASK LVDS

Figure 5: LAC configuration register

this is a 16 bits range read and write register; in fig. 5 the bits configuration is shown;

on the LAC board there are three trigger inputs: LVDS, NIM_P and NIM_R; it is possible to send a software trigger also; each trigger type can be activated or deactivated from the configuration register;

the polarity of trigger inputs can be set from positive to negative by switching three dedicated bits (the value "1" of the bit means negative polarity); the meaning of the configuration register bits is:

- read/write→
  - bit 0: MASK LVDS = 1 → LVDS enabled;
  - bit 1: MASK NIM_P = 1 → NIM_P enabled;
  - bit 2: MASK NIM_R = 1 → NIM_R enabled;
  - bit 3: MASK SOFT = 1 → software trigger enabled;
  - bit 4: POL LVDS = 1 → negative polarity;
  - bit 5: POL NIM_P = 1 → negative polarity;
  - bit 6: POL NIM_R = 1 → negative polarity;
- write only→bit 15: TRIG CLR →writing "1" in this bit forces the reset of the trigger status register; the hardware reset also this bit to zero.

8. LAC CONF - ADDRESS 0x0007

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAP START	VAB RESET	0	0	0	0	0	0	0	0	0	0	0	0	TEST VAB	TEST USB

Figure 6: LAC configuration register

this is a 16 bits range read and write register (see the configuration in fig. 6):

- read/write
  TEST USB (bit 0) = 1 means that LAC board is in test mode; in this configuration is possible to read from and write into the LAC ram memory;
  TEST VAB (bit 1) = 1 means that the VA boards are in test mode; in this configuration the VABs send to LAC a predefined value for each channel;
- write only
  VAB RESET (bit 14) = 1 starts the digital reset procedure of the VA boards; the hardware resets the bit to zero;
  MAP START (bit 15) = 1 starts the MAPPING procedure of the VAB connected to LAC; the hardware resets the bit to zero.

9. VAB MAP - ADDRESS 0x0008

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAB15 PRES	VAB14 PRES	VAB13 PRES	VAB12 PRES	VAB11 PRES	VAB10 PRES	VAB9 PRES	VAB8 PRES	VAB7 PRES	VAB6 PRES	VAB5 PRES	VAB4 PRES	VAB3 PRES	VAB2 PRES	VAB1 PRES	VAB0 PRES

Figure 7: VAB map register

this is a 16 bits read only register (see fig. 7); the LAC has 16 connectors for VABs; when the MAP procedure is invoked (by setting bit 15 in the

LAC CONF register), the value of i-th register bit switch to "1" if a VA board is plugged into the i-th LAC connectors.

10. HOLD VALUE SET - ADDRESS 0x0009

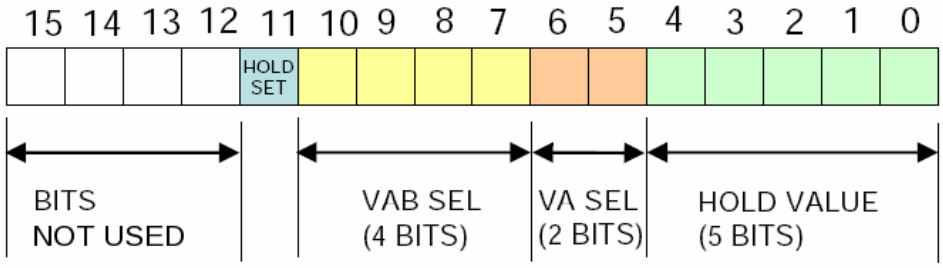

Figure 8: hold value set register

this is a 16 bits write only register; it permits to set the hold time of each VA chips of all VA boards; the configuration of the register bits (see fig. 8) follows:

- HOLD VALUE (bits from 0 to 4): these bits permit to insert a hold time value from zero to 3.2 μs in 32 steps of 100 μs (e.g. the binary value 1010 corresponds to 1 μs);
- VA SEL (bits from 5 to 6): these bits permits to select which of two VA chips of a VAB will be configured (01 means first chip, 10 second one, 11 both);
- VAB SEL (bits from 7 to 10): these bits permits to select the VA board;
- HOLD SET (bit 11): this bit must set be set "1" to change the hold value;

the remaining bits are not used.

3 Software Installation

The communication between the LAC and a PC is based on a QuickUSB module, developed by Bitwise Systems. To communicate with QuickUSB module it is necessary to install a precompiled library.

From the Bitwise Sistems website it is possible to download the file "QuickUSB_Library_v2.11.2.zip". This file permits to install the quickusb library under MS Windows XP Operating System (to install a password is necessary; this password is "endpoint").

After Windows installation it is possible to extract the quickusb library for Linux system under the folder "Program Files/Bitwise Systems/QuickUSB/Library/Linux/"; there are two versions of the library: the static version named libquickusb.a and the dynamic version named libquickusb.so.2.0.0. To install this library under a Linux Operating System it is sufficient to copy (like root

super user) the library in the folder named "/usr/lib". If you install the dynamic version you have also to make a symbolic link: libquickusb.so.2.0.0 —> libquickusb.so.

Under Windows XP OS, in the folder named "Program Files/Bitwise Systems/QuickUSB/Samples/Linux/QuickUsbDiagQT/", it is possible to extract also the include file "QuickUSB.h" that contain the declarations of all functions of the quickusb library. These two files (libquickusb.so.2.0.0 and QuickUSB.h) must be included in every program (written in C/C++ language) that you will use to communicate with a QuickUSB module. In the same folder there are also the two files (CQuickUSB.h and CQuickUSB.cpp); these file include the implementation of a C++ class, based on the quickusb library, optimized to communicate with a QuickUSB module.

The quickusb library and the CQuickUSB class have been tested successfully under Fedora Linux OS from version 3 up to version 10.

4 LAC initialization procedure and data taking

When the LAC is turned on, an initialization procedure must be performed. A schematic resume of this procedure (with some examples based on CQuickUsb class) follows:

1. Run Initialization

   (a) Search of QuickUSB devices:
   char dlist[128];
   CQuickUsb::FindModules(dlist,128);
   build a list of all QuickUSB modules connected to the host;
   the devices are named recursively "QUSB-0"..."QUSB-1"... etc. ;

   (b) Initialization of device:
   fQUSB = new CQuickUsb("QUSB-0");
   fQUSB->Open();
   fQUSB->WriteSetting(SETTING_WORDWIDE, 1);
   fQUSB->WriteSetting(SETTING_DATAADDRESS, 0x8000);
   the QuickUSB module has certain settings that control the behavior of the module;
   the SETTING_WORDWIDE must be set to "1"
   usually the SETTING_DATAADDRESS is set to 0x8000 (don't increment address bus mode [1]);

   (c) VAB mapping:
   unsigned short commandAddress = (LAC_CONF & 0x7FFF);
   unsigned char commandData = new unsigned char[2];*
   commandData[0] = 0x8000 & 0xff;
   commandData[1] = ((0x8000 & 0xff00) >> 8);
   fQUSB->WriteCommand(commandAddress, commandData, 2);
   the map of VAB boards connected to LAC is performed by writing the value "0x8000" into LAC CONF register;

(d) Setting of the Hold Time:
normally the hold time should not be set. It is set to 2.0 μs by default at power-on. Please see the hold time scan procedure to test the trigger timing;

(e) Setting of the Trigger Mask:
the trigger mask must be activated by writing into the TRIG CONF register;
example → normal DAQ with physics trigger:
commandAddress = (TRIG_ CONF & 0x7FFF);
commandData[0] = 0x8022 & 0xff;
commandData[1] = ((0x8022 & 0xff00) >> 8);
fQUSB->WriteCommand(commandAddress, commandData, 2);
in this example NIM_P input is activated with negative polarity; a trigger clear procedure is invoked by writing "1" into bit 16 of the register;

(f) Setting of High Voltage of silicon sensors:
the HV value of all silicon sensors can be set by writing into HIGH VOLTAGE VALUE register;
(VERY IMPORTANT!!!) the max permitted value is 700 DAC;
it is advisable to set the desired value by ramping up the high voltage with steps of 20 DAC with a time interval of two seconds between each step;

2. Data Taking:

(a) Initialization:
when a data taking run starts, a binary file to store the data is created;
the number of physics events to store can be set;

(b) Event Loop:
a polling procedure on the TRIGGER STATUS register is performed:

```
unsigned short length =2;
commandData[0]=0;
while(usr_ interrupt)
{
      commandData[0]=0;
        fQUSB->ReadCommand(TRIGGERS_ STATUS, commandData,
&length);
      if((commandData[0] & 0x20)!=0) break;
}
```

the bit 5 of register is checked; when an event (bit 5 = 1) is ready to be transferred the polling breaks; the event can be read from the LAC ram memory and stored in a buffer:
fQUSB->ReadData(Buffer, &length_ buf);
and then written in the binary file;
then the Trigger clear procedure must be invoked (to be able the LAC to receive a new trigger):

commandAddress = (TRIG_CONF & 0x7FFF);
commandData[0] = 0x8022 & 0xff;
commandData[1] = ((0x8022 & 0xff00) >> 8);
fQUSB->WriteCommand(commandAddress, commandData, 2);
after that the Event Loop can continue;

3. Run Termination:

   when the desired number of physics events are stored (or when an external interrupt occurs), the Event Loop ends;

   the binary file is then closed.

5 Special Runs

1. Calibration of the Hold Time:

   in order to optimally set the Hold-Time with respect to the trigger, a Hold-Time scan should be performed (e.g in the interval 1.8 - 2.2 μs);

   the hold time of each VAB boards is set by writing into HOLD VALUE SET register;

   an example of the setting of the hold time is shown:

   commandAddress = (HOLD_VALUE_SET & 0x7FFF);

   commandData[0] = 0x08c4 & 0xff;

   commandData[1] = ((0x08c4 & 0xff00) >> 8);

   fQUSB->WriteCommand(commandAddress, commandData, 2);

   in this example an hold time of 2 μs is set in the two VA chips of second VAB boards;

2. Pedestal Runs:

   sometimes it will be necessary to take some pedestal runs (e.g. runs with random trigger) in order to obtain the channel pedestal values;

   to do that the trigger mask must be modified:

   commandAddress = (TRIG_CONF & 0x7FFF);

   commandData[0] = 0x8044 & 0xff;

   commandData[1] = ((0x8044 & 0xff00) >> 8);

   fQUSB->WriteCommand(commandAddress, commandData, 2);

   in this example NIM_P input is deactivated and the NIM_R input is then activated (with negative polarity) in order to permit to LAC to receive a random trigger signal;

References

[1] ***"QuickUSB User Guide"***, Bitwise system, available on line at web address: http://www.quickusb.com/.