



Beam Loss Monitor

User Manual and Specifications



Table of contents

© Copyright Instrumentation Technologies 2015

Written by Peter Leban | August 23, 2019

Approved by Marko Praznik | September 13, 2019

Version 1.8

Revision history

Written by Peter Leban | January 21, 2016

Revision description Document creation

Version 1.0

Written by Peter Leban | February 12, 2016

Revision description Added Appendix A

Version 1.1

Written by Peter Leban | June 10, 2016

Revision description Corrected input signal range, SA history

Version 1.2

Written by Peter Leban | April 11, 2016

Revision description Updated with new functionalities

Version 1.2

Written by Peter Leban | July 17, 2017

Revision description Updated with new functionalities

Version 1.4

Written by Peter Leban | December 13, 2017

Revision description Updated with new functionalities

Version 1.6

Written by Peter Leban | January 15, 2019

Revision description Updated with new functionalities

Version 1.61

Written by Peter Leban | August 23, 2019

Revision description Updated with new functionalities

Version 1.8

No part of this document may be reproduced or stored on any medium without the written permission of Instrumentation Technologies.

Edition

First edition, January 2016

Printed in Slovenia

Instrumentation Technologies, Velika Pot 22

SI-5250 Solkan, Slovenia

Assistance

You can rely on our Technical support. Our core team consists of skilled engineers with full knowledge of the systems. We will help you with hardware, software, or system integration issues throughout the product's life cycle.

Contact us

E-mail: support@i-tech.si

Phone: +386 5 335 2600

Fax: +386 5 335 2601

Technologies licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Document dependencies

Table of content

1. Scope and purpose	6
2. General information	7
2.1 Safety symbols and terms	7
2.2 Abbreviations	7
2.3 List of supplied items	8
2.4 Hardware installation requirements	8
2.5 Maintenance	9
2.6 Transportation and storage	9
2.7 Warranty	9
3. Libera BLM interfaces	11
3.1 Front panel	11
3.2 Back panel	11
3.3 Platform management	12
3.3.1 Diagnostics and Thermal protection	12
3.3.2 Status & Error logging	12
3.3.3 Power supply	12
3.3.4 Network settings	12
3.3.5 Connecting to Libera BLM	13
3.3.5.1 SSH connection	13
3.3.5.2 USB console connection (Linux)	13
3.3.5.3 USB console connection (Windows)	14
3.3.6 Connecting external data storage to the USB port	14
3.3.7 Upgrade	14
3.3.8 Booting options	15
4. Libera BLM usage	17
4.1 Power up	17
4.2 Basic settings	17
4.2.1 Setting the network environment	17
4.2.2 Setting up the NTP server and local timezone	19
4.3 Application daemon	19
4.4 Instrument control and signal acquisition	19
4.4.1 MCI registry	20
4.4.2 Factory defaults and custom configuration	20
4.4.3 Accessing the registry	21
4.4.4 Libera BLM Parameters	23
4.5 Multi user and multi application simultaneous access	27
5. Libera BLM functionalities	28
5.1 Signal processing	28
5.2 Data paths	29
5.2.1 ADC buffer	29
5.2.2 ADC buffer, synthetic	30
5.2.3 ADC integrated buffer	31
5.2.4 SUM buffer	31
5.2.4.1 ADC offset and abs()	32
5.2.4.2 ADC mask	33
5.2.4.3 SUM decimation	34

5.2.4.4 Postmortem buffer	34
5.2.5 AVG buffer	36
5.2.6 SA stream	37
5.2.7 Counter stream	37
5.2.7.1 Normal counting mode.....	39
5.2.7.2 Differential counting mode.....	40
5.2.7.3 Selectable detection masks for the Counter stream.....	41
5.2.7.4 Default settings for different counting modes	42
5.2.8 Coincidence counter stream	44
5.2.9 Read out limitations.....	46
5.3 Input gain control.....	46
5.4 Input impedance selection.....	47
5.5 Triggering options.....	48
5.6 Trigger delay.....	49
5.7 Channel line delay.....	49
5.8 Sampling clock	50
5.8.1 T0 PLL.....	51
5.9 Event stream	51
5.10 BLD power supply and gain control.....	52
5.11 Calibration functionality	52
6. Software.....	55
6.1 General overview.....	55
6.2 Hardware components	56
6.3 Software components.....	56
6.4 Booting	57
6.4.1 Zynq boot ROM.....	57
6.4.2 FSBL	57
6.4.3 U-BOOT	57
6.4.4 Linux, device tree and RAM disk image	58
6.4.5 Libera BLM daemon.....	58
6.5 Building.....	58
6.5.1 U-BOOT	59
6.5.2 Linux and device tree	59
6.5.3 boost library.....	60
6.5.4 Libera BLM daemon.....	60
6.5.5 Boot image	60
7. Specifications	61
7.1 General specifications	61
7.2 Hardware inputs	61
7.3 Hardware outputs	62
7.4 Enclosure.....	63
8. Appendix A: Beam Loss Detector.....	64
8.1 Specifications	64
8.2 Hardware interfaces	64
8.3 Installation	65

Index of figures

Figure 1: Horizontal installation of 2 instruments	8
Figure 2: Vertical installation of 8 instruments.....	9
Figure 3: Front panel	11
Figure 4: Back panel.....	11
Figure 5: Boot options with or without the SD card.	15
Figure 6: Signal processing scheme.	28
Figure 7: ADC mask parameters.	33
Figure 8: Postmortem buffer.....	34
Figure 9: PM buffer data readout.	35
Figure 10: Counter stream branch in details.	38
Figure 11: Normal counting mode, 1 threshold, dead time 1.	39
Figure 12: Normal counting mode, 2 thresholds, dead time 1.	40
Figure 13: Differential counting mode.	40
Figure 14: Details of differential threshold detection.	41
Figure 15: Detection masks for the Counter stream.	42
Figure 16: Coincidence detection window.....	44
Figure 17: Correlation check setting.....	45
Figure 18: GUI screenshot for coincidence counting settings.....	46
Figure 19: Description of "trigger" events.....	49
Figure 20: Libera BLM software structure	55
Figure 21: Hardware components.	56
Figure 22: Software components.	56
Figure 23: Hardware output circuit.	63
Figure 24: Instrument's dimensions.....	63
Figure 25: Interfaces on the beam loss detector.....	65
Figure 26: Beam loss detector.....	65

Index of tables

Table 1: Abbreviations.....	7
Table 2: List of supplied items	8
Table 3: Factory and custom configuration files.....	21
Table 4: Libera BLM application parameters.....	23
Table 5: Data paths and data rate properties.....	29
Table 6: Default settings for counting modes.	43
Table 7: Input gain control.	47
Table 8: ADC full scale vs attenuation.	47
Table 9: Event ID description.	51
Table 10: General specifications	61
Table 11. Hardware inputs specifications.....	61
Table 12: BLD connector pin description.	62
Table 13. Dip switch configuration.	62
Table 14: Specifications for the beam loss detector.	64

1. Scope and purpose


This document contains information about the beam loss monitor processor instrument, Libera BLM. In combination with compatible beam loss detector, it forms a beam loss monitoring system. Libera BLM basis on a 4 channel 125 MHz digitizer and implements a signal processing customized for the storage rings. It features a power supply and gain control connections to the photo-multipliers installed in the beam loss detectors. Libera BLM includes a state-of-the-art digital electronics with low power consumption. It is fully compatible with today's standard of Power-over-Ethernet.

This document contains important information about the instrument and its usage. If in any case in doubt, please consult support@i-tech.si for technical support.

2. General information

This chapter contains general information about the instrument such as safety symbols and terms used in the documentation, abbreviations, etc.

2.1 Safety symbols and terms

The  symbol on the instrument indicates that the user should refer to the operating instructions located in the manual.

The **CAUTION** heading used in this manual explains hazards that could damage the instrument. Such damage may invalidate the warranty.

The **NOTE** heading used in this manual gives important explanations on the usage to avoid misunderstandings.

2.2 Abbreviations

There are several abbreviations used in this document. Please use Table 1 for their description.

Table 1: Abbreviations.

Abbreviation	Description
ADC	Analog to digital converter
BLD	Beam loss detector
BLM	Beam loss monitor
CPU	Central Processing Unit
DAC	Digital to Analog converter
DDR	Double Data Rate (computer memory)
DHCP	Dynamic Host Configuration Protocol
FPGA	Field Programmable Gate Array
GbE	Gigabit Ethernet
IP	Internet Protocol address
LED	Light Emitting Diode
NTP	Network Time Protocol
OS	Operating System
PC	Personal Computer
PLL	Phase Locked Loop
RMS	Root Mean Square
SD	Secure Digital (memory card)

Abbreviation	Description
SDRAM	Synchronous Dynamic Random Access Memory
SMA	Sub Miniature version A (connector)
SoC	System on Chip
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

2.3 List of supplied items

The instrument delivered to users is accompanied with item listed in Table 2.

Table 2: List of supplied items

Accessories included
Instrument
Warranty
Test record
Printed documentation: User Manual and Specifications document
Electronic media with software and documentation
Micro SD card

2.4 Hardware installation requirements

The instrument can operate as a desktop unit as it generates no noise and does not require forced cooling. Optionally, the instrument can be mounted in a standard 19" rack in horizontal or vertical orientation. It is possible to install 2 instruments in parallel horizontally (Figure 1) or up to 8 instruments in parallel vertically (Figure 2). There must be 2" spacing next to top panel of the instrument – the panel with heat sink - to ensure air circulation for adequate heat dissipation.



Figure 1: Horizontal installation of 2 instruments



Figure 2: Vertical installation of 8 instruments

Note: Rack mounting accessories are available as an option.

2.5 Maintenance

The instrument is designed in such way that it does not require special attention or maintenance actions by the user. There are no moving parts (e.g. fans) or air filters that would require periodic replacement. Software is installed in the SD memory card (in the instrument) or located in central location (server) and loaded at boot time.

2.6 Transportation and storage

Use original package for any kind of transportation or long-term storage. For shipments via mail, the package must be marked as fragile.

2.7 Warranty

The manufacturer guarantees its customers that the products it supplies are free from defects in materials and workmanship for a period of 2 years.

This warranty shall not apply to any defect, failure or damage caused by improper use or inadequate maintenance and care. Instrumentation Technologies shall not be obliged to provide service under this warranty to repair damage resulting from maintenance by personnel other than Instrumentation Technologies representatives.

This warranty is limited to the repair and, if necessary, the replacement of the instrument based solely on the decision of Instrumentation Technologies. Each instrument is subjected to a quality test. Any early failures are detected by this method.

3. Libera BLM interfaces

3.1 Front panel

The instrument's front panel contains 4 interfaces:

- USB used for data storage.
- RJ-45 for 1000Base-T Cu GbE connection: Main communication channel with the control system. Two LED diodes indicate link and activity status.
- Micro-B USB for USB connection: Used for serial console and is mainly used for development and service purpose.
- MicroSD memory card slot: Place for microSD card with software and OS.

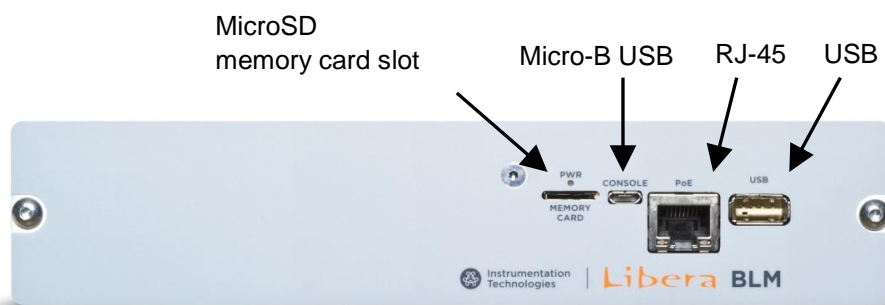


Figure 3: Front panel

In addition to interfaces, there is a green LED that indicates the unit is powered up and a company's logo and instrument's name.

3.2 Back panel

The back panel contains the following interfaces:

- 4x SMA for signal inputs: Used for connection to BLDs.
- 3x LEMO EPL.00.250.NTN for trigger inputs: T2 is used for triggering the acquisition.
- 4x RJ-25 6p6c: Power supply and gain control for the photosensors.
- Dip switch panel: Used to set the power supply and gain control voltage limit for the photosensors.

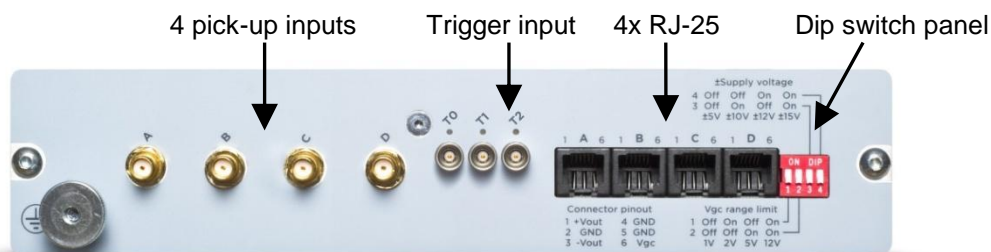


Figure 4: Back panel

3.3 Platform management

3.3.1 Diagnostics and Thermal protection

Temperature damage protection is enabled when the temperature of the Zynq SoC exceeds 86°C, in this case the instrument will automatically go in shutdown. The instrument will automatically start again when the temperature of Zynq will go below 67°C. Temperature and voltages of Zynq SoC can be read out with the following command:

```
root@libera:~# sensors
Zynq temperature [C]: 53
Zynq voltage vccint [mV]: 1011
Zynq voltage vccaux [mV]: 1011
Zynq voltage vccbram [mV]: 1011
Zynq voltage vccpint [mV]: 1010
Zynq voltage vccpaux [mV]: 1806
Zynq voltage vccoddr [mV]: 1513
Zynq voltage vrefp [mV]: 0
Zynq voltage vrefn [mV]: 0
```

The output is a temperature value expressed in °C and voltages expressed in millivolts. Nominal value for voltages vccint, vccaux, vccbram, vccpint is 1 V (± 50 mV), for voltage vccpaux is 1.8 V (± 100 mV) and for voltage vccoddr is 1.5 V (± 100 mV). Voltages vrefp and vrefn are not used.

3.3.2 Status & Error logging

General status of the daemon, warnings and errors are written in the libera-blm.log file. The log file is located in /tmp directory and is not persistent, what means that after the unit power cycle the content of the file is lost. The entries in the Status & Error log file have one of the following prefixes that enable easy searching inside the file:

- Info
- Warning
- Error
- Critical

3.3.3 Power supply

The instrument is powered by PoE (48 V, max 15 W). The instrument complies with electrical safety standards.

3.3.4 Network settings

The network interface is configured for DHCP by default. If no DHCP is present the DHCP client will timeout after 20 seconds and set the network interface to default IP address (192.168.1.100). Static IP assignment is also supported. If the connection is broken and re-established, automatic Ethernet configuration/connection recovery is supported. The OS provides a ping response.

3.3.5 Connecting to Libera BLM

3.3.5.1 SSH connection

If no DHCP is present the DHCP client will set the network interface to default IP address (192.168.1.100). To connect via SSH connection, do:

```
ssh root@192.168.1.100
```

The SSH login information is:

- Username: root
- Password: Jungle

3.3.5.2 USB console connection (Linux)

The serial console is supported via USB connection on the instrument's front panel. Use console connection to observe complete boot procedure.

Linux based OS (Ubuntu) users can use »minicom« to use console connection. Once the USB cable is connected to PC and Libera BLM, open a terminal window and check USB devices (look for »Future Technology Devices«).

```
user@ubuntu:~$ lsusb | grep Future
Bus 005 Device 002: ID 0403:6015 Future Technology Devices International, Ltd
```

The tty USB0 device must also be listed in /dev.

```
user@ubuntu:~$ ll /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 2013-11-27 08:09 /dev/ttyUSB0
```

Some older Ubuntu versions (10.04, 11.04) do not recognize the USB device. To fix this, issue the following command in the terminal window.

```
sudo modprobe ftdi_sio vendor=0x0403 product=0x6015
```

To load the driver automatically at boot, add this line to /etc/modules file.

```
ftdi_sio vendor=0x0403 product=0x6015
```

If all above works, run »minicom« as shown below.

```
user@ubuntu:~$ minicom -D /dev/ttyUSB0
```

Serial console will open with login prompt. Use username and password to login.

Minicom connection properties:

- Baud rate: 115200
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: Hardware

3.3.5.3 USB console connection (Windows)

To use console connection under Windows OS, download and install the FTD driver from <http://www.ftdichip.com/Drivers/VCP.htm> website. After installation, a new COM port will be present in Device Manager. Use Hyperterminal or another terminal utility (e.g. Putty) to connect to Libera BLM.

Connection properties:

- Baud rate: 115200
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: Hardware

This connection was verified under WindowsXP (32-bit), Windows7 (64-bit) and on Windows8.1 (64-bit).

3.3.6 Connecting external data storage to the USB port

External data storage (USB key) can be connected to the USB input. You can check the presence of the USB key with the following command:

```
root@libera:~# fdisk -l | grep /dev/sda1
```

You have to mount the disk, in order to start using it. Before disk removal, it is strongly recommended to unmount the device.

```
root@libera:~# mount /dev/sda1 /mnt
root@libera:~#
root@libera:~# umount /mnt
```

3.3.7 Upgrade

All vital software (OS, application) is stored in one place that is in the SD memory card or in the image on TFTP server. If an upgrade is required, it can be done by exchanging the SD memory card with the new image or by upgrading the installed packages. The instrument must be powered OFF during memory card exchange. In case of a network boot, the upgrade is done by changing the target image on the TFTP server. The upgrade can be performed by using the “opkg” package manager. Use the following command to check the options for “opkg”.

```
root@libera:~# opkg -h
```

Upgrade procedure is detailed in the readme file on the installation media, provided together with new unit. Currently installed packages can be checked in the following way (package naming and versions vary).

```
root@libera:~# rw
root@libera:~# opkg list-installed
libera-base-tools - 3.0-418+r22822+helium
libera-base3.0 - 3.0-418+r22822+helium
libera-blm - 1.0-67+r2059+helium
libera-blm-generic - 1.0-67+r2059+helium
libera-cli - 3.0-418+r22822+helium
libera-fpga-blm-generic-itc7020 - 1.0-36+r2060
libera-kernel - 3.0-418+r22822+helium
libera-mci3.0 - 3.0-418+r22822+helium
root@libera:~#
```

Complete list of installed packages may be different in case of EPICS or TANGO interface installations.

NOTE: File system has to be set in read/write mode to list the packages.

3.3.8 Booting options

Libera BLM supports two boot methods:

- Local boot
- Network boot

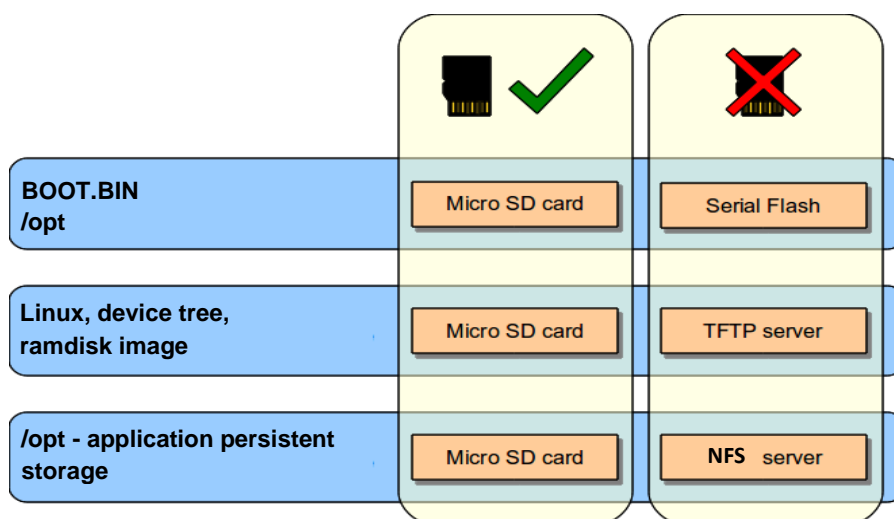


Figure 5: Boot options with or without the SD card.

The type of boot is defined by the SD card presence at power cycle (not at reboot) and defines both the device/location of the OS files and the device/location for the /opt application persistent storage (Figure 5). Thus, only these two combinations are supported:

- Local boot: SD card boot + /opt on SD card
- Network boot: TFTP boot + /opt on NFS share

If the network boot is selected (SD card absent), the TFTP boot filename is defined by the "filename" DHCP parameter, and the NFS share to be mounted to /opt is defined by the "root-path" DHCP option:

```
host libera-001 {
    hardware ethernet 00:0A:35:00:01:22;
    fixed-address 192.168.1.150;          # Quasi-static IP. Optional (default = dynamic IP
address).
    filename "spark-legacy/ucommands_tftp"; # TFTP boot file. Optional (default =
zynq/ucommands_tftp).
    server-name "192.168.1.1";          # TFTP boot server. Optional (default = DHCP
replier).
    option root-path "192.168.1.1:/nfs/opt"; # NFS share for /opt. Optional: If not present, SD
card is mounted on /opt.
    option host-name "libera-001";
}
```

This way, the OS TFTP file locations, as well as the /opt NFS share selection are fully configurable in a centralized DHCP server configuration file. Nothing prevents the user to configure the DHCP server to point the NFS share and the OS boot files to the same directory, creating the same directory structure as on the SD card.

NOTE: If NFS share is set to different location than the TFTP directory, the NFS share must contain all OS files (same as TFTP directory).

NOTE: TANGO users should assign the DHCP to set the hostname correctly.

The boot file "ucommands_tftp" includes the path to OS boot files (Linux kernel, devicetree, ramdisk). If the "filename" DHCP parameter is modified from the default, most probably you want to modify the "ucommands_tftp" file as well:

```
# On the development host:
server@server:~$ dd if=ucommands_tftp of=commands_tftp bs=72 skip=1
server@server:~$ cp commands_tftp commands_tftp-custom
server@server:~$ editor commands_tftp-custom
server@server:~$ sudo apt-get install u-boot-tools
server@server:~$ mkimage -T script -C none -n 'Commands' -d commands_tftp-custom
ucommands_tftp-custom
```


4. Libera BLM usage

4.1 Power up

The instrument is powered up after connected to the GbE network switch. The switch must support PoE functionality. The LED diode on the front panel indicates the power-on status. Boot procedure starts and lasts typically around 30 seconds. Login is possible after approximately 30 seconds after power-up.

The boot procedure starts with U-boot and continues with OS image loading. The boot-up procedure consists of the following stages:

- Power ON
- Boot load selection (SD card / TFTP)
- U-boot loading
- Linux OS and file system load (SD card / TFTP)
- Application start (includes FPGA load)
- Configuration set (locally or by the Control System)

4.2 Basic settings

Once connected to Libera BLM, the user can easily modify basic settings. With the root username, the user has full access to complete software structure. The user “root” has the default password “Jungle”.

CAUTION: Be careful when modifying basic settings as this may result in a crash of the complete Libera BLM.

Note that the file system is **read-only by default**; this was done to avoid unnecessary writing to flash memory. One must set the system to read-write before writing. We propose that after the writing procedure, the file system is changed back to read-only

```
root@libera:~# ro
root@libera:~#
root@libera:~# rw
```

4.2.1 Setting the network environment

The default network configuration is DHCP with a static fallback IP of 192.168.1.100, if no reply is received from the DHCP server within the timeout period of ~60s. Static network configuration takes precedence over DHCP if the **/nvram/etc/network/eth0_static** file exists and is executable. In this case, the eth0_static script is executed. A template for the eth0_static file can be found in **/etc/network/templates/nvram/etc/network/eth0_static**. In order to edit the eth0_static, open it with “nano” editor and edit parameters under “# Static network configuration”.

```
root@libera: cat /etc/network/templates/nvram/etc/network/eth0_static
#!/bin/sh
#
# Libera static network script
#
# Edit & copy to /nvram/etc/network/eth0_static for static network configuration.
#

# Static network configuration
ADDRESS=192.168.1.2
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS=192.168.1.1
HOSTNAME=libera-001

# NFS share for /opt. Leave blank for SD card.
NFS=

echo "Setting static network configuration:"
echo "  ADDRESS: ${ADDRESS}"
echo "  NETMASK: ${NETMASK}"
echo "  GATEWAY: ${GATEWAY}"
echo "  DNS: ${DNS}"
ifconfig eth0 ${ADDRESS} netmask ${NETMASK} up
route add default gw ${GATEWAY}
echo "nameserver ${DNS}" > /etc/resolv.conf

# Wait for the eth0 link-up
sleep 8

# Mount /opt application persistent storage
/etc/init.d/opt_mount ${NFS}
```

If user wants to use the settings from **/etc/network/templates/nvram/etc/network/eth0_static**, the script has to be manually copied to the **/nvram/etc/network/eth0_static**. Please follow the below procedure and reboot the unit. After the reboot, you can connect to the unit with the defined static IP address.

```
root@libera:~# mount -w -o remount /dev/mtdblock1 /nvram
root@libera:~#
root@libera:~# cd /nvram/
root@libera:/nvram# mkdir -p /nvram/etc/network
root@libera:/nvram# cd
root@libera:~# cp /etc/network/templates/nvram/etc/network/eth0_static
/nvram/etc/network/
root@libera:~# chmod a+x /nvram/etc/network/eth0_static
root@libera:~#
root@libera:~# mount -r -o remount /dev/mtdblock1 /nvram
root@libera:~#
```

The **eth0_static** can be simply deleted from the nvram to enable DHCP again.

```
root@libera:~# mount -w -o remount /dev/mtdblock1 /nvram
root@libera:~# rm /nvram/etc/network/eth0_static
root@libera:~# mount -r -o remount /dev/mtdblock1 /nvram
root@libera:~#
```

4.2.2 Setting up the NTP server and local timezone

To set up a custom NTP server IP and a local timezone where the instrument is used, follow these instructions.

In your Linux PC

1. Make a custom timezone file. Example for China Standard Time:

```
echo "Zone CST +8:00 - CST" > CST.zone
```

2. Compile and install to somewhere locally:

```
zic CST.zone -d ~/.zoneinfo
```

3. Copy the .zoneinfo/CST file to your instrument

```
scp ~/.zoneinfo/CST root@192.168.1.100:/opt/
```

In your instrument

4. Edit the /opt/etc/init.d/S99local file to add these lines:

```
rw
sed -i /etc/init.d/S49ntpd -e 's/pool.ntp.org/192.168.1.1/g'
cp /opt/CST /etc/localtime
/etc/init.d/S49ntpd restart
exit 0
```

Note, that '192.168.1.1' is an example for the IP address of the (local) NTP server. If global NTP server is used, omit this line.

4.3 Application daemon

The application daemon consists of the FPGA design and application. To check the application daemon version, do:

```
root@libera:~# libera-ireg version
version: 1.0-67-r2059
```

To start, stop or restart the application daemon, please do the following:

```
root@libera:~# /opt/etc/init.d/S50libera-blm start
root@libera:~# /opt/etc/init.d/S50libera-blm stop
root@libera:~# /opt/etc/init.d/S50libera-blm restart
```

4.4 Instrument control and signal acquisition

This chapter describes the configuration parameters that offer a wide range of usage possibilities of Libera BLM. The way of changing these parameters through the libera-ireg command line tool is described.

4.4.1 MCI registry

MCI registry is a tree-like structure that provides the user with information about Libera BLM status and configuration and at the same time enables to change the configuration parameters and to access the acquired data.

The registry tree has the following functionalities:

- external representation of data structure
- transparent access to local and remote nodes
- network interface for easy remote access
- notification mechanism

Each configuration parameter and / or status information is represented in the registry tree as a node. Individual nodes are identified by names (similarly as directories and files). The tree nodes are populated by the instrument software dynamically, depending on the hardware setup and type of the instrument. Each of the nodes can contain a value of a predefined type (integer number, floating point number, text, enum – represented in textual format, none – serves only to define structure).

There are two types of nodes, namely local and remote nodes. Local nodes are those that contain information inside the calling process. Remote nodes are just links to the local nodes inside another process (also on another computer). MCI is implemented as the user (client) interface to access the nodes in the registry tree. It provides uniform access to all nodes in the registry, thus the location (local, remote) and node provider implementation specifics are hidden from the user. MCI interface is used by the end clients like command line interface (CLI), graphical user interface (GUI) and control system adaptors (EPICS, Tango...).

The notification mechanism is implemented in the MCI registry. It gives user the possibility to subscribe to receive notifications when the node value changes or other conditions occur. There are two types of notifications which are sent when:

- value of the node is changed
- value of the node is out of range

The available data streams (signals) are enumerated in the registry and defined by instrument application software. Data stream classes provide access to different types of signals. Classes simplify the processing of the signals (either in the embedded computer or by the client).

4.4.2 Factory defaults and custom configuration

When the application daemon starts up, custom configuration is loaded to the vital parameters. If no custom configuration exists, factory defaults will be loaded instead. Some application parameters are essential for correct instrument's operation and must not be modified by the user runtime but only when application daemon is stopped. For this reasons, application parameters are marked with different properties:

- Readable: Runtime modification not allowed. Value can be modified only through custom configuration file.
- Writable: Runtime modification is allowed.
- Persistent: Parameter's value can be saved to customer configuration file.
- Hidden: Parameter is not shown in the registry tree.

After virgin installation (or when custom configuration file does not exist yet), custom configuration must be created in order to modify parameters marked as readable&persistent. Custom configuration file can be created:

- Automatically (on application exit)
- Manually (on user's request)

Automatic custom configuration file creation is enabled/disabled as shown below:

```
root@libera:~# libera-ireg system.persistence.save_on_exit=true
root@libera:~# libera-ireg system.persistence.save_on_exit=false
```

Custom configuration file is created manually as shown below. Application daemon can be running normally.

```
root@libera:~# libera-ireg system.persistence.save{}
system.persistence.save{}: done
```

Table 3 shows file locations. If specific parameters have to be modified in the custom configuration file (if it is readable only), modification must be done to the original file location (/nvram/cfg/). /nvram must be mounted as read/write first. After modifications are done, remount back to read-only.

```
root@libera:~# mount -o remount,rw /nvram/
root@libera:~# nano /nvram/cfg/libera-blm.xml
root@libera:~# mount -o remount,ro /nvram/
```

Table 3: Factory and custom configuration files.

Configuration type	File location
Factory default	/opt/libera/sbin/cfg/libera-blm.xml
Custom	/nvram/cfg/libera-blm.xml (/var/opt/libera/cfg is a symbolic link to /nvram/cfg)

In case the custom configuration file gets corrupted or contains parameters that cause application daemon crash, factory defaults can be used. To use factory defaults:

- Stop the application daemon (if it runs)
- Remount the /nvram/ to read/write
- Erase /nvram/cfg/* files
- Remount the /nvram/ to read-only
- Start the application daemon. It will load factory default settings.
- Create custom configuration file (automatically or manually). Edit the parameters of interest.

4.4.3 Accessing the registry

Registry can be used by the instrument application software (local access) or remotely (network access). MCI provides the interface between the registry tree and the user. Based on this interface a command line utility called libera-ireg is implemented as a tool to easily explore the registry tree and as an example of using the registry tree. Commands that libera-ireg utility offers can be seen by typing:

```
root@libera:~# libera-ireg help

Available commands:
  info          [Print info of the registry nodes.]
  dump          [Dump the registry tree.]
  access        [Access values of the registry nodes.]
  listen        [Listen to the notifications on the registry nodes.]
  signal        [Print out signal data.]

General HELP:
- Tree nodes are addressed with its 'Full Path' which
  is visible in the output of the 'info' command.
- The 'FullPath' is composed by node names divided
  only by '.' (dot)
- Setting values can be done only with this syntax:
  [nodeFullPath]=[value]
  the '=' is mandatory without any space
```

The command **info** displays the detailed information of the selected registry node(s).

```
root@libera:~# libera-ireg help info
. . .
```

The command **dump** is used to display the whole registry tree or sub-tree.

```
root@libera:~# libera-ireg help dump
. . .
root@libera:~# libera-ireg dump -l 2
. . .
```

The command **access** is used to display or change the value of the selected registry node(s). The same effect is achieved if the word access is omitted. The value of the writable registry node is set by typing the command the full path of the node, the = sign and the new value of the node.

```
root@libera:~# libera-ireg help access
. . .
```

The command **listen** is used to wait for the notifications from the selected node(s). Typically, the notification is send when node's value changes.

```
root@libera:~# libera-ireg help listen
. . .
```

One part of the registry tree is dedicated to data buffers and streams. The command **signal** is used to capture and display the acquired data. All attributes of the data path can be read from the registry tree by using the command "dump" or "info".

```
root@libera:~# libera-ireg help signal
. . .
root@libera:~# libera-ireg signal application.signals.adc -s 10 -bEvent
. . .
```

4.4.4 Libera BLM Parameters

In this chapter, the application specific parameters are described. Libera BLM configuration and control is done through the parameters listed in below table. Properties column specifies the valid setting range, type and parameter properties:

- r : readable
- w : writable
- p : persistent
- c : constant

Table 4: Libera BLM application parameters

Parameter name / path	Description	Properties
application. triggers.t2.delay	Sets the delay on the external trigger arrival. The delay is set in steps of ADC samples.	Setting range [0, 32767] ULong r,w,p
application. events.t2.count	Read the number of T2 triggers since reboot (or application restart)	ULongLong r
application. clock_info.adc_frequency	Information about sampling clock	Setting range [10000000, 125000000] ULong r,c
application.clock_info. pll_ref.frequency	Reference clock on T0 for PLL locking	Double r
application.clock_info. pll_ref.decimation	Division factor between the ADC frequency and reference clock	ULong r,p
application. hk.termination.{A, B, C, D}	Selection of input impedance (50 Ohm / 1 MOhm).	Setting range: 0 = 1MOhm, 1 = 50Ohm ULong r,w,p
application. hk.attenuation.{A, B, C, D}	Sets the variable attenuator to selected value (0...31)	Setting range [0, 31] ULong r,w,p
application.signal_processing. max_adc.{A, B, C, D}	Displays the second max value on ADC data	Read only
application.signal_processing. adc_mask.offset	Sets the offset for the ADC mask from first ADC sample	Setting range [0, 4096] ULong r,w,p
application.signal_processing. adc_mask.window	Sets the ADC mask window length	Setting range [1, 4096] ULong r,w,p
application.signal_processing. adc_offset.{A, B, C, D}	Sets the offset in the ADC data (counts)	Setting range [-8192, 8191] Long r,w,p
application.signal_processing. abs_enabled.{A, B, C, D}	Enable/disable abs() operation on ADC data	Setting range [0,1] Bool r,w,p

Parameter name / path	Description	Properties
application.signal_processing.decimation.sum	Sets the decimation factor from ADC to SUM	Setting range [16, 4096] ULong r,w,p
application.signal_processing.decimation.sum_mode	Select SUM mode (integration or averaging)	Setting range: 0 = integrating, 1 = averaging ULong r,w,p
application.signal_processing.decimation.sumdec_periods	Set number of SUM_DEC periods for ADC integrated buffer	Setting range: [2,50] ULong r,w,p
application.signal_processing.decimation.avg.n	Sets the decimation factor from SUM to AVG	Setting range [1,18] ULong r,w,p
application.signal_processing.decimation.avg.val	Displays the actual decimation factor from SUM to AVG	Calculated from avg.n; 2^n , n=[1,18]
application.signal_processing.decimation.sa.n	Sets the decimation factor from SUM to SA	Setting range [16,26] ULong r,w,p
application.signal_processing.decimation.sa.val	Displays the actual decimation factor from SUM to SA	Calculated from sa.n; 2^n , n=[16,26]
application.signal_processing.decimation.t0_interval	Sets the decimation factor for the ADC masks in the Counter stream	Setting range [16,4096] ULong r,w,p
application.signal_processing.trig_source	Sets the acquisition trigger source	Setting range: 0 = Off, 1 = External, 2 = Threshold ULong r,w,p
application.signal_processing.threshold.loss_count	Sets the threshold for counter stream	Setting range [-8192,8191] but dynamically adjusted ULong r,w,p
application.signal_processing.threshold.loss_count_high	Sets the threshold reset for counter stream	Setting range [-8192,8191] but dynamically adjusted ULong r,w,p
application.signal_processing.threshold.counter_dead_time	Sets the dead time value for counter stream	Setting range [0,32] Long r,w,p
application.signal_processing.threshold.counter_diff	Sets the threshold for differential counting mode	Setting range [1,32767] but dynamically adjusted Long r,w,p
application.signal_processing.counting_mode	Switch between the counting modes	0 ... differential mode; 1 ... normal mode r,w,p

Parameter name / path	Description	Properties
application.signal_processing.coincidence.mask.A.{B,C,D}	Coincidence detection setting: Master = channel A Set B,C,D to 1 to enable comparison. Set B,C,D to 0 to ignore.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.mask.B.{A,C,D}	Coincidence detection setting: Master = channel B Set A,C,D to 1 to enable comparison. Set A,C,D to 0 to ignore.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.mask.C.{A,B,D}	Coincidence detection setting: Master = channel C Set A,B,D to 1 to enable comparison. Set A,B,D to 0 to ignore.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.mask.D.{A,B,C}	Coincidence detection setting: Master = channel D Set A,B,D to 1 to enable comparison. Set A,B,D to 0 to ignore.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.mask.D.{A,B,C}	Coincidence detection setting: Master = channel D Set A,B,D to 1 to enable comparison. Set A,B,D to 0 to ignore.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.correlation_check.A.{B,C,D}	Coincidence detection setting: Master = channel A Set 1 to use B,C,D »as is« for comparison. Set 0 to use $\bar{B}, \bar{C}, \bar{D}$ for comparison.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.correlation_check.B.{A,C,D}	Coincidence detection setting: Master = channel B Set 1 to use A,C,D »as is« for comparison. Set 0 to use $\bar{A}, \bar{C}, \bar{D}$ for comparison.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.correlation_check.C.{A,B,D}	Coincidence detection setting: Master = channel C Set 1 to use A,B,D »as is« for comparison. Set 0 to use $\bar{A}, \bar{B}, \bar{D}$ for comparison.	Setting range [0, 1] ULong r,w,p

Parameter name / path	Description	Properties
application.signal_processing.coincidence.correlation_check.D.{A,B,C}	Coincidence detection setting: Master = channel D Set 1 to use A,B,C »as is« for comparison. Set 0 to use $\bar{A}, \bar{B}, \bar{C}$ for comparison.	Setting range [0, 1] ULong r,w,p
application.signal_processing.coincidence.window.{A,B,C,D}	Sets the window for coincidence detection after a loss event in the master channel (A,B,C,D)	Setting range [1, 16] ULong r,w,p
application.signal_processing.adc_counter_mask1.offset	Offset setting for Mask1 for the Counter stream	Setting range [0, 4096] ULong r,w,p
application.signal_processing.adc_counter_mask1.window	Window setting for Mask1 for the Counter stream	Setting range [1, 4096] ULong r,w,p
application.signal_processing.adc_counter_mask2.offset	Offset setting for Mask2 for the Counter stream	Setting range [0, 4096] ULong r,w,p
application.signal_processing.adc_counter_mask2.window	Window setting for Mask2 for the Counter stream	Setting range [1, 4096] ULong r,w,p
application.signal_processing.threshold.autotrig.edge_detection	Select the raising/falling edge for autotriggering	Setting range: 0 = falling edge, 1 = raising edge ULong r,w,p
application.signal_processing.threshold.autotrig.{A,B,C,D}	Sets the threshold for auto triggering the signal processing.	Setting range [-8192,8191] ULong r,w,p
application.detector.power_supply	Information about power supply voltage at RJ-25 output	Setting range [5V,10V,12V,15V] by dip switch Enumeration r
application.detector.vgc.limit	Information about gain control voltage limit at RJ-25 output	Setting range [1V,2V,5V,12V] by dip switch Enumeration r
application.detector.vgc.output.{A,B,C,D}	Sets the value (in Volts) of the gain control voltage at RJ-25 output.	Setting range [0,12], depending on the power supply set and gain control limitation Double r,w
application.signals.counter.data_rate	Sets the data rate (in samples/second) for counter stream.	Setting range [1,10] ULong r,w
application.signal_processing.pm.enable	Enables the PM detection	Executable node

Parameter name / path	Description	Properties
application.signal_processing. pm.running	Shows the status of PM (running=true means data is being written)	Setting range: automatic by the application Bool r
application.signal_processing. pm.data_available	Reports if PM data is available for read	Setting range: automatic by the application Bool r
application.signal_processing. pm.adc_mask.offset	Sets the offset for the ADC mask from first ADC sample (PM only)	Setting range [0, 4096] ULong r,w,p
application.signal_processing. pm.adc_mask.window	Sets the ADC mask window length (PM only)	Setting range [1, 4096] ULong r,w,p
application.signal_processing. pm.adc_offset.{A, B, C, D}	Sets the offset in the ADC data (counts) (PM only)	Setting range [-8192, 8191] Long r,w,p
application.signal_processing. pm.abs_enabled.{A, B, C, D}	Enable/disable abs() operation on ADC data (PM only)	Setting range [0,1] Bool r,w,p
application.signal_processing. pm.decimation	Sets the decimation factor from ADC to SUM (PM only)	Setting range [16, 4096] ULong r,w,p
application.signal_processing. pm.sum_mode	Select SUM mode (integration or averaging) (PM only)	Setting range: 0 = integrating, 1 = averaging ULong r,w,p
application.triggers.t1.delay	Set the Postmortem trigger delay	Setting range: [0, 131071] ULong r,w,p
application.calibration. enabled	Enable/disable calibration	Setting range [0,1] Bool r,w,p
application.calibration. calculated_coeff.{A, B, C, D}	Read current calibration coefficients	Setting range: automatic by the application Double r
application.calibration. bldCalib.{A, B, C, D}	Set the BLDCalib correction coefficients	Setting range [1,10000] ULong r,w,p
application.calibration. vgc.{A, B, C, D}	Set the Vgc reference values for G	Setting range [0,12] Double r,w,p
application.calibration. g.{A, B, C, D}	Set the Vgc correction coefficients	Setting range [0,65535] Double r,w,p

4.5 Multi user and multi application simultaneous access

Digital signal processing and the software structure allow unprecedented possibilities. It is possible to access all data paths at the same time. One user can monitor the streamed and buffered data at the same time, another user from another computer can do studies with Raw ADC data. The only limitation is the capacity of the network, which limits the frequency of the acquisition of big buffers.

5. Libera BLM functionalities

5.1 Signal processing

Analog signal conditioning sets each of the input channels (A,B,C,D) the input termination (50 Ω or 1 M Ω) and the gain (programmable attenuation).

Digital signal conditioning and processing can be split to two major branches:

- Triggered data (buffers)
- Continuously processed data (counter streams)

Input signals are sampled at (PLL-controlled) ADC sampling clock. Upon an acquisition trigger event, ADC samples in each of the four processing chains are summed or averaged (over user-defined number of ADC samples) and stored to the SUM and AVG buffers. Raw ADC data is stored as well. The application daemon manages all parameters (yellow) and does buffer read-out (orange). See Figure 6.

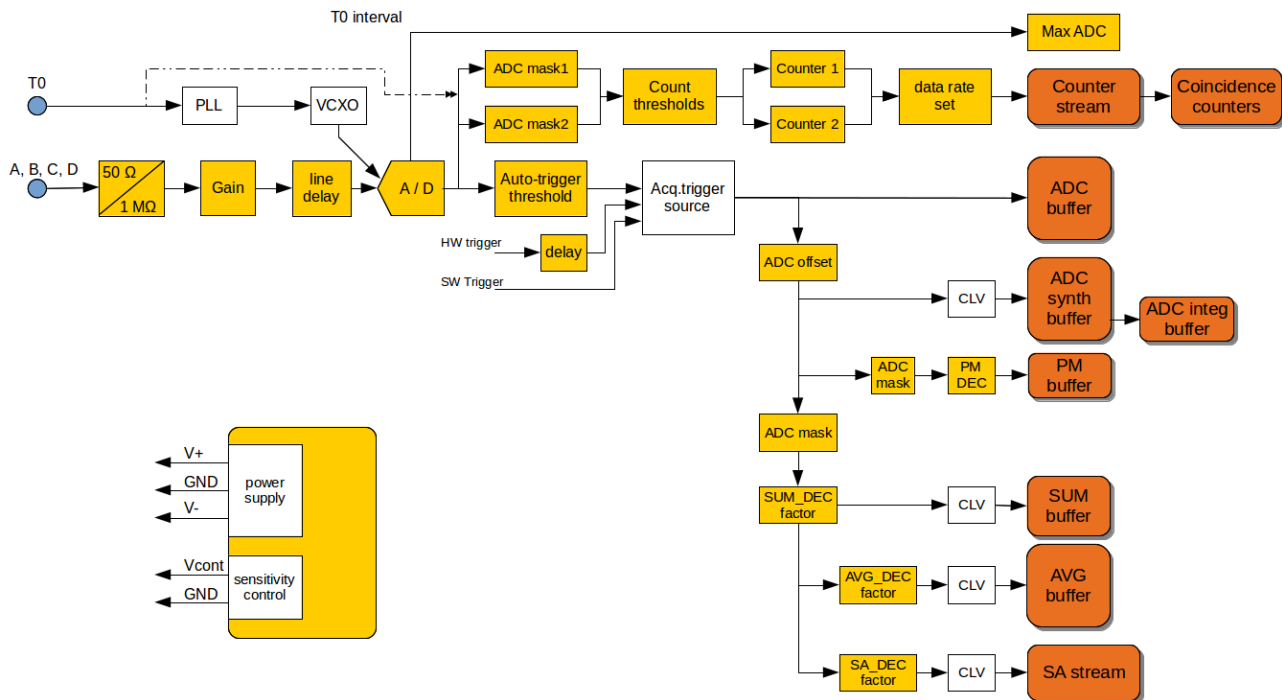


Figure 6: Signal processing scheme.

In addition to a triggered data type, the ADC data is continuously processed and checked for a “count” threshold. The “count threshold” and a selection of other counting-related parameters specify the behaviour of the counter algorithm that increments the counters’ values. The “data rate set” block specifies the read interval (e.g. 0.1 second). Result is a ‘Counter stream’, shown in Figure 6 on top. Counter streams between A,B,C,D channels are used for coincidence beam-loss detection. Coincidental events are written to the coincidence counter stream.

Signal processing is specified by several decimation or averaging factors, and several ADC masks. All parameters are described in the next chapters in details.

5.2 Data paths

Figure 6 shows the signal processing and resulting data paths that are available to user for read-out. Properties of data paths are given in Table 5.

Table 5: Data paths and data rate properties.

Data path	Atom size per channel	Total buffer size	Number of data points per channel	Data rate [samples/second]
ADC buffer	2	64 MB	8 million	f_{ADC}
ADC buffer (synth)	2	131 kB	16384	f_{ADC}
ADC integrated	2	See explanation in Chapter 5.2.3		
SUM buffer	4	16 MB	1 million	$\frac{f_{ADC}}{SUM_DEC}$
PM buffer	4	16 MB	1 million	$\frac{f_{ADC}}{PM_DEC}$
AVG buffer	4	16 MB	1 million	$\frac{f_{ADC}}{SUM_DEC * AVG_DEC}$
SA stream	4	(stream)	1 (stream)	$\frac{f_{ADC}}{SUM_DEC * SA_DEC}$
Counter stream	8	(stream)	1 (stream)	$1 < f < 10$ (int) selectable
Coincidence counter stream	4	(stream)	1 (stream)	$1 < f < 10$ (int) selectable

5.2.1 ADC buffer

The ADC buffer contains four amplitudes (one from each input channel). Amplitudes are given with 14-bit resolution (= [0:16383] ADC counts). One atom consists of four 16-bit words. MSBs [15:14] are 0. The total buffer size is 64 MB and can accommodate up to 8,388,608 ADC samples per channel but is split to 8 segments.

NOTE: To fill the ADC buffer with fresh data, acquisition trigger event is required. See Chapter 5.5 for details about triggering options.

To read the data from the ADC buffer, see example below (switch '-v' provides metadata with signal data):

```

root@libera:~# libera-ireg signal application.signals.adc -s3 -bEvent
      138      192      134      57
      137      193      135      52
      139      193      132      57

root@libera:~# libera-ireg signal application.signals.adc -s3 -v
Signal data from node: cache.host-127-0-0-1:5679.app.application.signals.adc
LMT: 12569037763914, ABSPOS: 0, COUNTER: 239
      A      B      C      D
      138      192      134      57
      137      193      135      52
      139      193      132      57
Total number of atoms: 3

```

Example shows the acquisition example with '-bEvent' switch. Using this switch, the libera-ireg client will wait for the trigger event and then output data.

The ADC data is continuously monitored for the second peak value (Max ADC). The peak value is latched in a register that is read out through a software parameter. The value resets on read. The input signal can vary significantly, especially if the gain or input impedance is modified. To ensure the correct and fresh value is read, a continuous monitoring shall be put on this parameter (or at least 2 sequential reads).

5.2.2 ADC buffer, synthetic

The synthetic or treated ADC buffer contains raw data from the ADC buffer (Chapter 5.2.1) which is compensated for the ADC offset. Values for ADC offset are given for each channel individually. The synthetic ADC buffer can be read on trigger.

$A_{\text{synth}} = |A_{\text{raw}} - A_{\text{offset}}|$ (same applies for amplitudes B, C and D)

NOTE: The abs() function is applied if enabled in the relevant registry node.

A_{synth} saturates at ± 13 bit value (8191 / -8192).

To read the data from the ADC synth buffer, see example below (switch '-v' provides metadata with signal data):

```

root@libera:~# libera-ireg signal application.signals.adc_synth -s3 -bEvent
      95      67      93      0
      99      60      95     -2
      93      56      94      0

root@libera:~# libera-ireg signal application.signals.adc_synth -s3 -bEvent -v
Signal data from node: cache.host-127-0-0-1:5679.app.application.signals.adc_synth
LMT: 397325041723, ABSPOS: 0, COUNTER: 35080
      A      B      C      D
      101      60      96      5
      102      63      93      0
      105      65      94      0
Total number of atoms: 3

```

Buffer size is limited to 16384 samples.

5.2.3 ADC integrated buffer

Assuming the Libera BLM is used in a circular accelerator with a fixed revolution frequency, the integration of ADC data over several turns can reconstruct a filling pattern. This method is limited by the ADC sampling frequency, the revolution frequency and the actual filling pattern.

One turn is presented with SUM_DEC number of ADC samples. ADC data is read over N turns. Application will read B=SUM_DEC x N number of ADC samples and perform calculation (see example in this chapter) and output buffer 'A'.

Number of SUM_DEC periods for ADC integrated buffer is set with:

```
libera-ireg application.signal_processing.decimation.sumdec_periods
```

Maximum number of periods for processing is 50.

```
SUM_DEC = 16
N = 5  ← SUMDEC_PERIODS

length(B) = 16 * 5 = 160
length(A) = SUM_DEC = 16

A(1)=sum( B(1) + B(17) + B(33) + B(49) + B(65) )
A(2)=sum( B(2) + B(18) + B(34) + B(50) + B(66) )
...
A(16)=sum( B(16) + B(32) + B(48) + B(64) + B(80) )
```

NOTE: When reading the ADC integrated buffer, requested buffer length must match SUM_DEC exactly.

5.2.4 SUM buffer

The SUM buffer contains four amplitudes (one from each input channel). Amplitudes are given with 32-bit resolution. One atom consists of four 32-bit words. The total buffer size is 8 MB and can accommodate up to 1,048,576 ADC samples per channel.

The ADC data is integrated (summed) or averaged over a user-defined number of consecutive ADC samples. The user-defined number is specified as "SUM_DEC". Integration/averaging window can be further adjusted through the ADC mask.

Integration mode:
$$SUM\ amplitude(j) = \sum_{i=1}^W x_i$$

Averaging mode:
$$SUM\ amplitude(j) = \frac{1}{SUM_DEC} \sum_{i=1}^W x_i$$

where:

SUM amplitude (j) ... j-th data sample in SUM buffer

x_i ... i-th ADC sample from selected integration window

W ...ADC mask window

Selection between the integration (default) and averaging mode is done through a registry node:

```
root@libera:~# libera-ireg application.signal_processing.decimation.sum_mode
sum_mode=0
```

SUM mode can be set to:

0 ... integration mode (default)

1 ... averaging mode

To read the data from the SUM buffer, see example below (switch '-v' provides metadata with signal data):

```
root@libera:~# libera-ireg signal application.signals.sum -s3 -bEvent
2224      3094      2136      865
2243      3075      2140      849
2243      3074      2122      845
root@libera:~# libera-ireg signal application.signals.sum -s3 -v
Signal data from node: cache.host-127-0-0-1:5679.app.application.signals.sum
LMT: 12569037763914, ABSPOS: 0, COUNTER: 239
      A      B      C      D
2224      3094      2136      865
2243      3075      2140      849
2243      3074      2122      845
Total number of atoms: 3
```

NOTE: To fill the SUM buffer with fresh data, acquisition trigger event is required. See Chapter 5.5 for details about triggering options.

5.2.4.1 ADC offset and abs()

Offset on the ADC data is caused by many factors (PMT, ADC, etc.) and is inevitable. A beam loss is usually detected as a short (negative) pulse. Data in the SUM buffer is integrated or averaged over several ADC samples. Contribution of the beam loss can easily be negated by the ADC offset.

The ADC offset is a ± 13 bit value (8191 / -8192) that is subtracted from the raw ADC value. It is given for each channel independently.

$$A' = |A_{\text{raw}} - A_{\text{offset}}| \quad (\text{same applies for amplitudes B, C and D})$$

The ADC offset is applied in the ADC synth data and SUM data processing and auto-trigger and count thresholds.

The abs() function can be enabled or disabled independent for each channel.

Example shows how to read/set the ADC offset and enable/disable abs() function.


```

root@libera:~# libera-ireg dump application.signal_processing.adc_offset
adc_offset
  A=0
  B=0
  C=0
  D=112
root@libera:~# libera-ireg application.signal_processing.adc_offset.D=300
root@libera:~# libera-ireg application.signal_processing.adc_offset.D
application.signal_processing.adc_offset.D: 300

root@libera:~# libera-ireg dump application.signal_processing.abs_enabled
abs_enabled
  A=false
  B=false
  C=false
  D=false
root@libera:~# libera-ireg application.signal_processing.abs_enabled.D=true
root@libera:~# libera-ireg application.signal_processing.abs_enabled.D
application.signal_processing.abs_enabled.D: true

```

NOTE: The ADC offset is not applied to raw ADC data buffer.

5.2.4.2 ADC mask

ADC mask is applied to the ADC data that is going to the SUM buffer. It is specified by 2 parameters:

- window: number of ADC samples that will be used for SUM data processing
- offset: sets the window delay in ADC samples after the trigger

See Figure 7 for more detailed view.

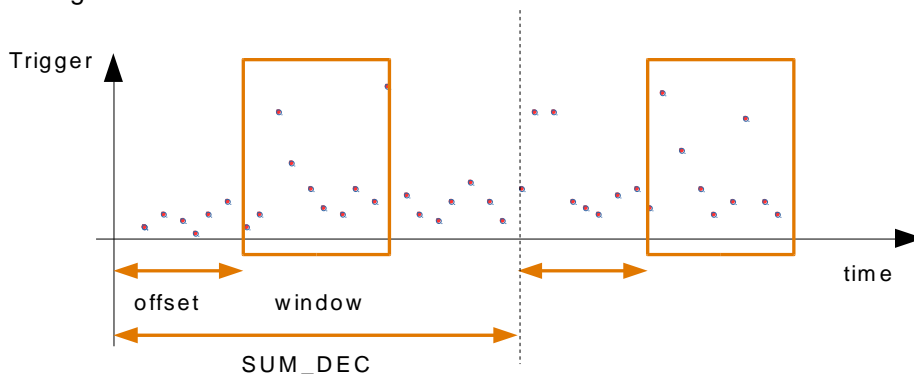


Figure 7: ADC mask parameters.

When applying signal processing parameters, make sure that:

$$(offset + window) < SUM_DEC$$

To read and modify the offset and window parameters, see example:

```

root@libera:~# libera-ireg dump application.signal_processing.adc_mask
adc_mask
  offset=0
  window=4096
root@libera:~# libera-ireg application.signal_processing.adc_mask.offset=2
root@libera:~# libera-ireg application.signal_processing.adc_mask.window=100
root@libera:~# libera-ireg dump application.signal_processing.adc_mask
adc_mask
  offset=2
  window=100
root@libera:~#

```

5.2.4.3 SUM decimation

Decimation factor from ADC to SUM data can be modified runtime. If modified during data processing, the data paths may get corrupted. The SUM decimation can obtain integer values from 16 to 4096.

To read and modify the SUM decimation, see example:

```

root@libera:~# libera-ireg application.signal_processing.decimation.sum
application.signal_processing.decimation.sum: 16
root@libera:~# libera-ireg application.signal_processing.decimation.sum=20
root@libera:~# libera-ireg application.signal_processing.decimation.sum
application.signal_processing.decimation.sum: 20
root@libera:~#

```

5.2.4.4 Postmortem buffer

The source data for the PM buffer is processed with same processing blocks as the data for the SUM buffer. Independent PM decimation, ADC offset and ADC mask can be set (functionalities described in Chapters 5.2.4.1 and 5.2.4.2).

The PM buffer is a circular buffer (FIFO) that is written continuously until the T1 trigger event. Then, the writing stops and the postmortem functionality is disabled automatically. The PM buffer contains a historical data prior the T1 event.

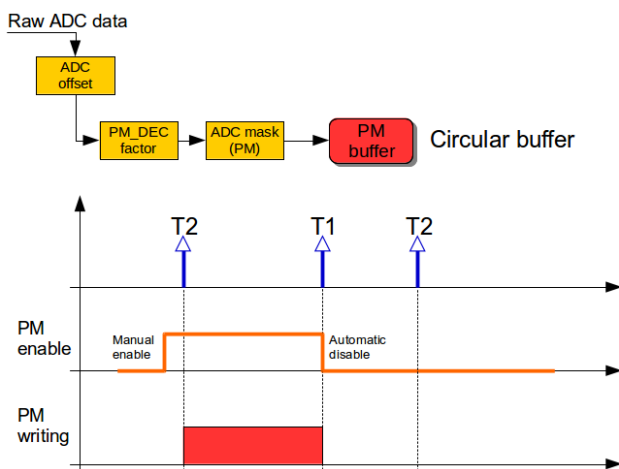


Figure 8: Postmortem buffer.

To re-enable the postmortem functionality, following 2 steps are required (also shown in Figure 8):

- Enable the postmortem functionality
- Wait for acquisition trigger (T2 or auto-trigger)
- PM buffer writing starts at location 0

PM writing is enabled with an executable registry node (just toggles one bit):

```
libera-ireg application.signal_processing.pm.enable{}
```

Software provides 2 variables that show the status of the PM buffer:

```
libera-ireg application.signal_processing.pm.running
application.signal_processing.pm.running: false

libera-ireg application.signal_processing.pm.data_available
application.signal_processing.pm.data_available: false
```

Once the PM is enabled and the acquisition trigger is fired, the `application.signal_processing.pm.running` shows status 'true'. This means the data is being written into the PM buffer. Once the T1 trigger is fired, writing stops. `application.signal_processing.pm.data_available` shows status 'true'.

If a new T1 trigger event arrives before the PM buffer is full (before it starts to overwrite), only the written part of the buffer is available for readout. See Figure 9 for details.

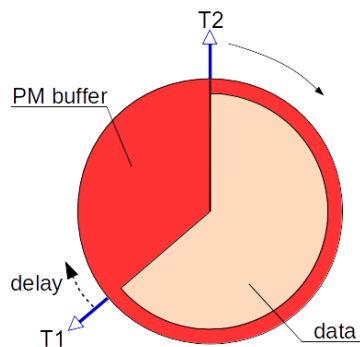


Figure 9: PM buffer data readout.

NOTE: Only the first acquisition trigger after the PM is enabled starts the writing process. Other acquisition triggers do not interrupt or re-synchronize the PM data!

The postmortem trigger can be delayed in order to record the behaviour of losses that happen after the critical event. Delay is set in ADC clock cycles. The absolute delay time depends on the current ADC sampling frequency. If the ADC sampling clock is set to 100 MHz, a delay setting of 100 reflects into 1 μ s. Setting range is from 0 to 131,071 (~1 ms).

To set the postmortem trigger delay, see example below (set delay to 10):

```
libera-ireg application.triggers.t1.delay=10
```

To read the data from the PM buffer, see example below:

```
root@libera:~# libera-ireg signal application.signals.pm -s10
      811      523      733      898
    1625    1042    1437    1810
    1605      979    1440    1768
```

Data is oriented backwards in time. Suggested read-out mode is '–bNow' (default).

5.2.5 AVG buffer

The AVG buffer contains four amplitudes (one from each input channel). Amplitudes are given with 32-bit resolution. One atom consists of four 32-bit words. The total buffer size is 8 MB and can accommodate up to 1,048,576 ADC samples per channel.

The SUM data is averaged over a user-defined number of consecutive SUM samples. The user-defined number is specified as "AVG_DEC".

$$AVG\ amplitude(j) = \frac{1}{AVG_DEC} \sum_{i=1}^{AVG_DEC} SUM_amplitude_i$$

where:

AVG amplitude (j) ... j-th data sample in AVG buffer

SUM amplitude (i) ... i-th data sample in the averaging window from SUM buffer

AVG_DEC ...averaging window length

NOTE: To fill the AVG buffer with fresh data, acquisition trigger event is required. See Chapter 5.5 for details about triggering options.

To read the data from the AVG buffer, see example below (switch '-v' provides metadata with signal data):

```
root@libera:~# libera-ireg signal application.signals.average -s3 -bEvent
      2227      3082      2120      856
      2220      3085      2112      856
      2217      3084      2117      859
root@libera:~# libera-ireg signal application.signals.average -s3 -v
Signal data from node: cache.host-127-0-0-1:5679.app.application.signals.average
LMT: 12569037763914, ABSPOS: 0, COUNTER: 239
      A          B          C          D
      2227      3082      2120      856
      2220      3085      2112      856
      2217      3084      2117      859
Total number of atoms: 3
```

Example shows the acquisition example with '-bEvent' switch. Using this switch, the libera-ireg client will wait for the trigger event and then output data.

5.2.6 SA stream

The SA stream contains four amplitudes (one from each input channel). Amplitudes are given with 32-bit resolution. One atom consists of four 32-bit words. The data in the SA stream is averaged over a user-defined number of consecutive SUM samples. The user-defined number is specified as "SA_DEC".

$$SA \text{ amplitude} = \frac{1}{SA_DEC} \sum_{i=1}^{SA_DEC} SUM_amplitude_i$$

where:

SA amplitude ... data sample in SA stream

SUM amplitude (i) ... i-th data sample in the averaging window from SUM buffer

SA_DEC ... averaging window length

NOTE: Acquisition trigger event restarts the SA stream. See Chapter 5.5 for details about triggering options.

To read the data from the SA stream, see example below:

```
root@libera:~# libera-ireg signal application.signals.sa -s3
      2486      3444      2377      941
      2486      3444      2373      942
      2483      3443      2370      943
```

Alternative SA data stream acquisition can be done from the SA history buffer. This is a special FIFO buffer that hold the last 8192 SA samples. It is possible to specify less samples at the read request. Data content is exactly the same as in the SA data stream. This buffer is particularly useful for TANGO ATK Panel client.

To read the SA history buffer, see example below:

```
root@libera:~# libera-ireg signal application.signals.sa.history -s3 -v
Signal data from node: cache.host-127-0-0-1:5679.app.application.signals.sa.history
LMT: 0, MT: 0, ABSPOS: 716
      A      B      C      D
      951      167      1027      885
      950      170      1027      888
      951      171      1026      887
Total number of atoms: 3
```

5.2.7 Counter stream

The ADC data is continuously provided to a counter stream branch. A T0 input (T0 interface on the instrument's back panel) is configured for a reference clock (it can be defined as shown in Chapter 5.8.1, relevant parameter is application.clock_info.pll_ref.frequency) that drives:

- Sampling clock (PLL)
- ADC masks on the Counter stream

Each T0 event (e.g. every 2 μ s) restarts the ADC masks that are applied to the incoming ADC data. See Figure 10. If default values are not modified, all ADC samples are processed with no interruption or exception.

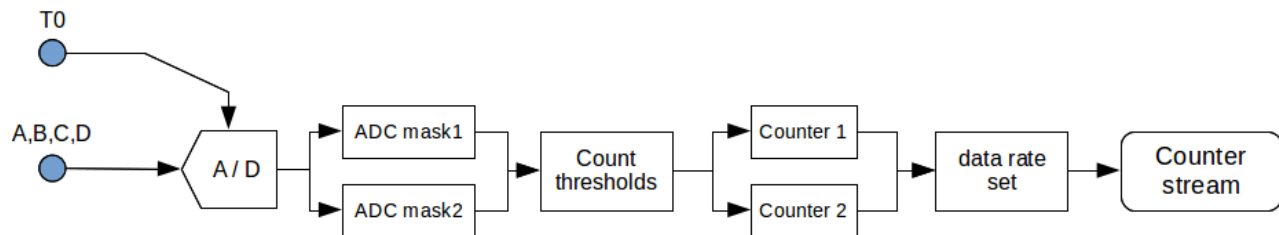


Figure 10: Counter stream branch in details.

NOTE: Counter stream is always processed on the raw ADC data. ADC offset and absolute function do not affect the Counter stream.

Loss counters are reset at specified data rate (configurable between 0.1 second to 1 second). The data rate parameter is common to all channels and defines a Counter stream data rate.

To read and modify the data rate parameter see example:

```

root@libera:~# libera-ireg application.signals.counter.data_rate
application.signals.counter.data_rate: 1
root@libera:~# libera-ireg application.signals.counter.data_rate=10
root@libera:~# libera-ireg application.signals.counter.data_rate
application.signals.counter.data_rate: 10
  
```

To read the data from the Counter stream, see example below:

```

root@libera:~# libera-ireg signal application.signals.counter -s3 -v
Signal data from node: cache.host-127-0-0-1:5679.app.application.signals.counter
A1      B1      C1      D1      A2      B2      C2      D2
743606  1335469  3688046  3685232  743606  1335469  3688046  3685232
731777  1310784  3686988  3684718  731777  1310784  3686988  3684718
741473  1347914  3687707  3684873  741473  1347914  3687707  3684873
  
```

The Counter stream contains 8 values (2x A, B, C, D). First 4 (A1, B1, C1, D1) are counters that belong to ADC mask 1, second 4 (A2, B2, C2, D2) are counters that belong to ADC mask 2. For ADC mask explanation, see Chapter 5.2.7.3.

NOTE: Typical signal from the photosensors is a negative pulse. Counting mode works on a signal falling edge only.

Counting of loss events can be generally split to 2 modes:

- Differential counting mode
- Normal counting mode

By default, counting mode is set to “normal”. When switching between the modes, some parameters are set automatically and blocked for modification. Check Chapter 5.2.7.1 and 5.2.7.2 for description of counting modes.

To select a counting mode, use registry node

```
libera-ireg application.signal_processing.counting_mode
```

where:

0 ... differential counting mode

1 ... normal counting mode

NOTE: The ADC buffer is acquired with reference to a T2 trigger. Data in the ADC buffer may not be aligned with a T0 trigger (it is off-set by a random value). Take this into consideration when setting up the ADC masks for the counting modes.

5.2.7.1 Normal counting mode

When an ADC sample drops below Threshold limit, this is considered as a loss event. A counter increments by 1. Detection algorithm then ignores the next (D) ADC samples. The “D” stands for a dead time. Signal must raise above the Threshold before a new loss event can be detected. See Figure 11 for details (dead time is 1).

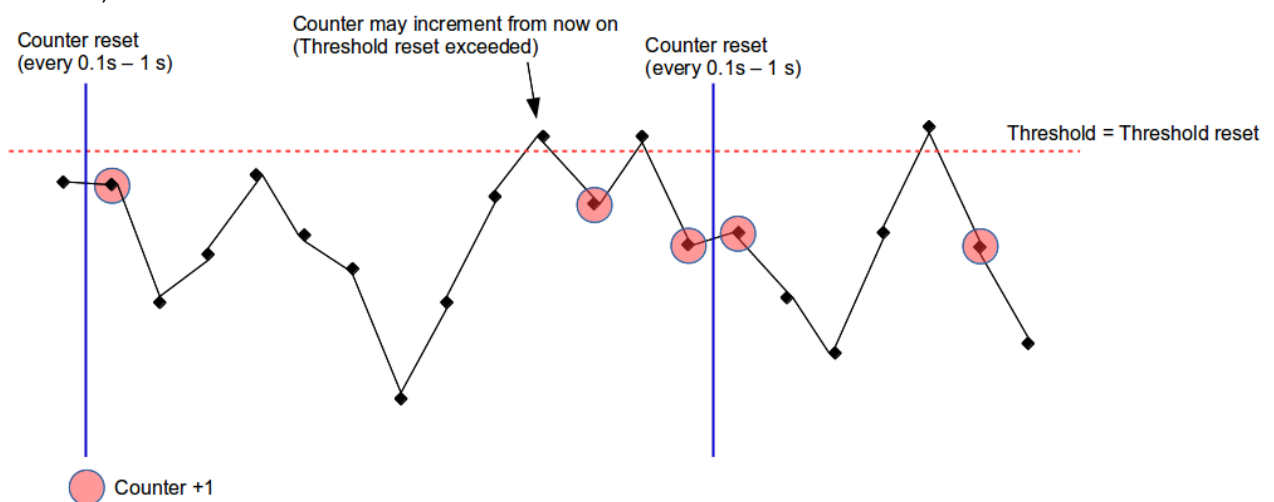


Figure 11: Normal counting mode, 1 threshold, dead time 1.

Counting mode can be enhanced with setting a second Threshold (“Threshold reset”). After a loss event, the signal must raise over the “Threshold reset” before a new loss event can be detected (Figure 12).

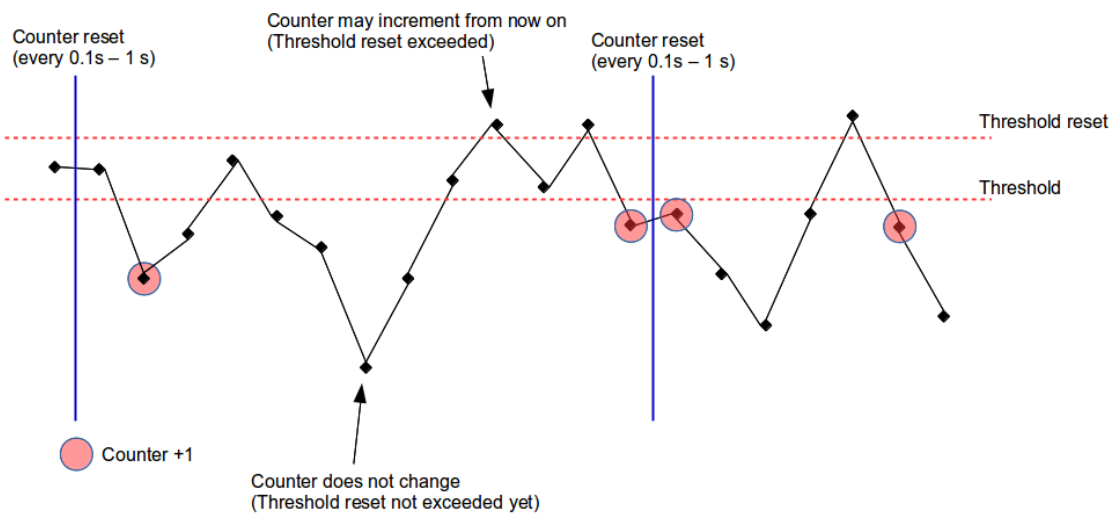


Figure 12: Normal counting mode, 2 thresholds, dead time 1.

NOTE: To use a single threshold value, both thresholds must be set to the same value.

5.2.7.2 Differential counting mode

Losses can be detected and counted based on the difference between the two neighbour samples. Counter will increment when the current sample's amplitude is lower enough (difference threshold) with reference to the previous sample's amplitude. See Figure 13 for examples.

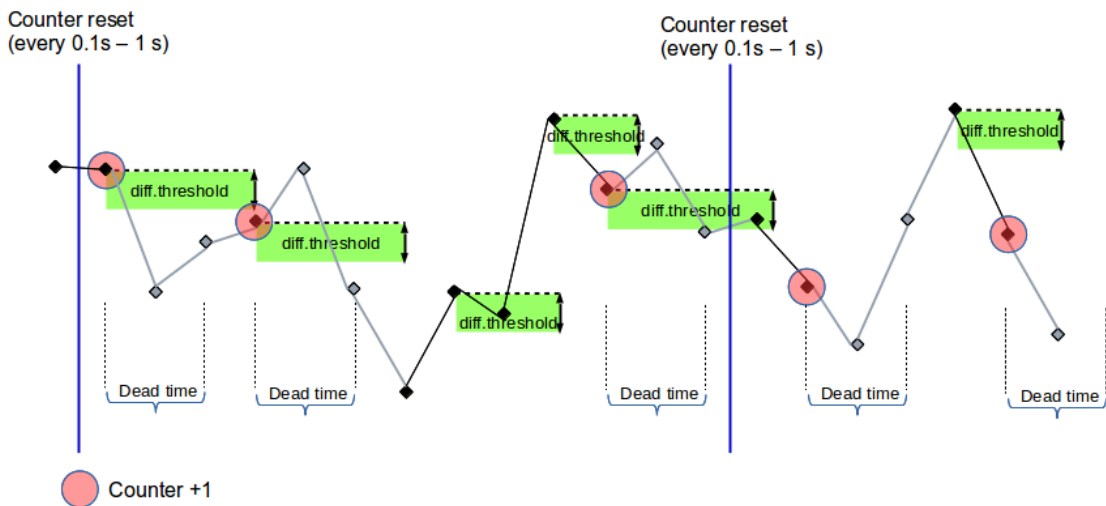


Figure 13: Differential counting mode.

Important facts and limitations:

- The first sample after the dead time is compared to the sample where loss was detected.

- Count does not increase when amplitude is falling continuously. After a detected loss, the amplitude must raise for at least 1 sample before difference threshold is checked again. See Figure 14 for details.

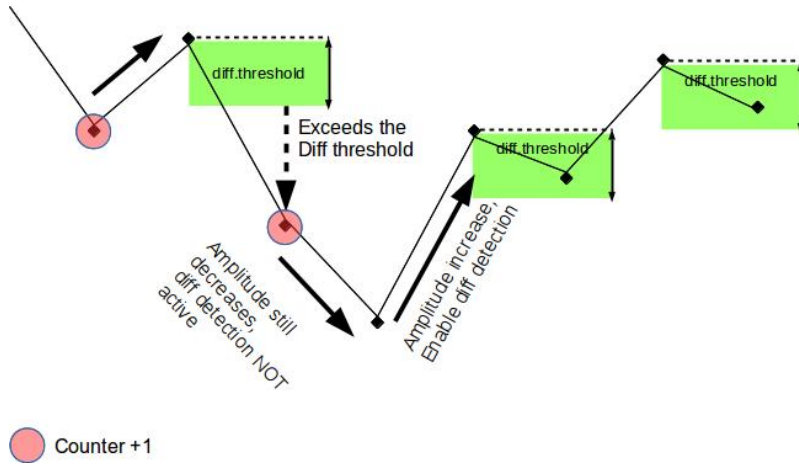


Figure 14: Details of differential threshold detection.

NOTE: Differential threshold parameter is given as a positive number in ADC counts.

5.2.7.3 Selectable detection masks for the Counter stream

Input data that is processed in the Counter stream is passing the user-specified processing window (t_0 interval). Within the processing window, there is an ADC mask, that selects only a portion of the processing window. The ADC mask is configured with an offset (starting point of the processing window) and a window (length in ADC samples). The rule for adjusting the offset and window parameters is:

$$offset + window \leq t_0_interval$$

The T0 events (Figure 15) are dictated by an incoming T0 trigger. In circular machines, this is usually a revolution or turn-by-turn clock. Processing window length (t_0 interval) shall match the number of ADC clocks between two T0 events. The t_0 interval becomes the total length of the ADC mask. An offset and window parameters are configured with reference to the T0 event.

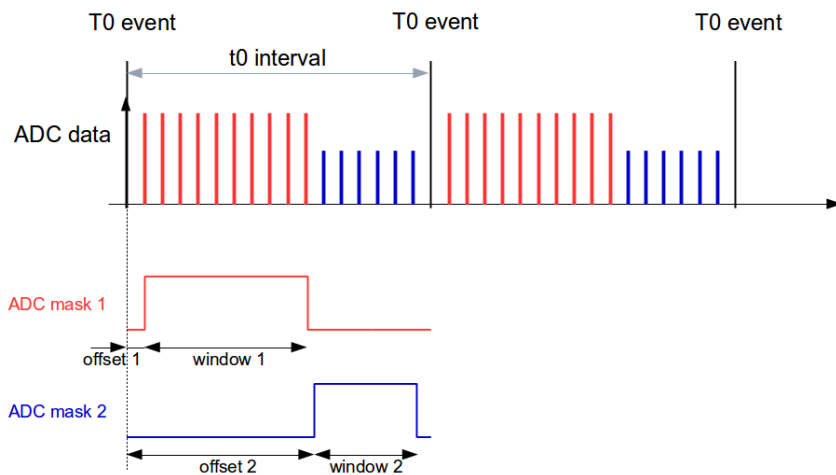


Figure 15: Detection masks for the Counter stream.

It is possible to set two independent ADC masks over the same t_0 interval. This yields two counter values that are pushed to the Counter stream.

Applying an ADC mask to the Counter stream allows the user to analyze losses only on a selected part of the fill pattern. If both masks are configured, one can read losses from two parts of the fill pattern that are analyzed independently.

Proper timing alignment between the T0 events, the sampling clock and ADC masks is guaranteed with a PLL (see Chapter 5.8.1). The t_0 interval is calculated with this equation (it is an integer value):

$$t_0 \text{ interval} = \frac{f_{ADC}}{f_{T0}}$$

Where:

f_{ADC} ... sampling clock (see Chapter 5.7)

f_{T0} ... frequency of the T0 trigger signal (parameter `pll_ref.frequency`, see Chapter 5.8.1)

If the t_0 interval is not aligned with the actual T0 event rate, the ADC mask will be continuously applied until the next T0 event. Default settings (no ADC masking) assure normal counting behaviour regardless the T0 signal connection.

5.2.7.4 Default settings for different counting modes

Loss detection algorithm can be customized for different counting modes. Table 6 contains default values for each of the modes. Settings can be further adjusted based on actual amplitudes and loss shape. Threshold-related parameters are given for each channel independently. t_0 interval and ADC masks' settings are common to all channels.

Table 6: Default settings for counting modes.

Parameter	Normal counting	Differential counting	Counting with ADC masks
Threshold (threshold.loss_count)	-200	8192 (fixed)	Same as selected counting mode
Threshold reset (threshold.loss_count_high)	-200	8192 (fixed)	
Dead time (threshold.counter_dead_time)	0	0	
Differential threshold (threshold.counter_diff)	32767 (fixed)	400	
T0 interval (decimation.t0_interval)	16	16	$\frac{f_{ADC}}{f_{T0}}$
ADC mask1 (adc_counter_mask1)	offset=0 window=16	offset=0 window=16	Based on the filling pattern
ADC mask2 (adc_counter_mask2)	offset=0 window=16	offset=0 window=16	

Parameters are located under the signal_processing registry node at different locations. See complete paths in the printout below. See Table 4 for full details of each parameter and their full path.

```

root@libera:~# libera-ireg dump
application.signal_processing.{decimation.t0_interval,adc_counter_mask1,adc_counter_ma
sk2}
t0_interval=17
adc_counter_mask1
  offset=0
  window=16
adc_counter_mask2
  offset=0
  window=16

```

```

root@libera:~# libera-ireg dump application.signal_processing.threshold
threshold
  loss_count
    A=8192
    B=8192
    C=8192
    D=8192
  loss_count_high
    A=8192
    B=8192
    C=8192
    D=8192
  counter_dead_time
    A=0
    B=0
    C=0
    D=0
  counter_diff
    A=400
    B=400
    C=400
    D=400

```

5.2.8 Coincidence counter stream

Losses detected by the counting algorithm can be correlated between the channels A, B, C and D. Simplistically said, if a loss event happens on two, three or all four channels at (almost) the same time, this is considered as a 'coincidence loss event'. A detection window starts after the master channel reports the loss event. If loss events are reported by other channels within the detection window, a coincidence count for the master channel increments by 1 (see Figure 16). Input data to coincidence detection scheme is a Counter stream.

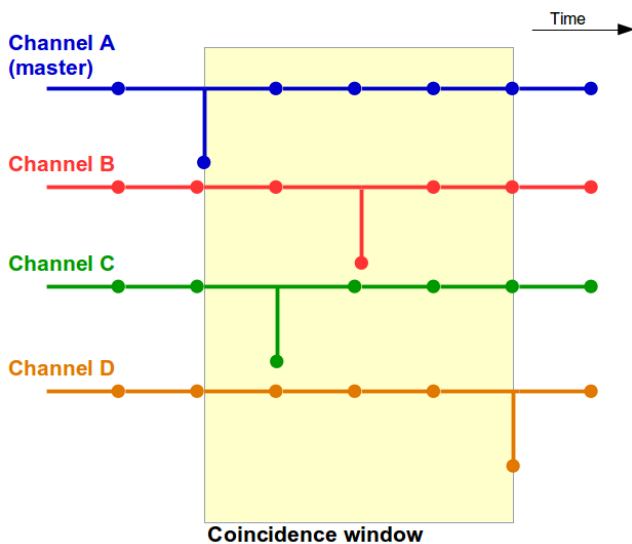


Figure 16: Coincidence detection window.

The coincidence counter stream contains 4 values (A,B,C,D). Value »A« contains the number of coincidence loss events when channel A was a master and was correlated with B, C and D. For other values in the Counter stream, channels B, C and D are the master channels compared to other 3:

Master	Compared with
A	B, C, D
B	A, C, D
C	A, B, D
D	A, B, C

Correlation between a master channel and other channels can be configured to:

- Use / ignore a channel
- Real / inverted channel

See Figure 17 for details. In this example, channel A is a master channel.

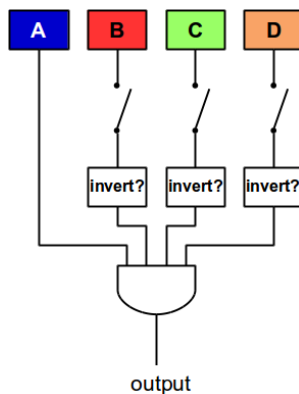


Figure 17: Correlation check setting.

Settings for the coincidence detection are located under registry node:

```
libera-ireg dump application.signal_processing.coincidence
```

Selection of which channel is compared (or not) is set with parameters under registry node:

```
libera-ireg dump application.signal_processing.coincidence.mask
```

Selection of which channel is inverted for comparison is set with parameters under registry node:

```
libera-ireg dump application.signal_processing.coincidence.correlation_check
```

There are numerous sub-nodes. Please refer to Table 4 for description of each node.

NOTE: Coincidence detection uses the loss events from a Counter stream processed with the first ADC mask (see Chapter 5.2.7.3).

Settings for the coincidence counting are shown in Figure 18. This figure shows the graphical interface (EDM panel for EPICS) where coincidence detection is set up:

- Channel A is compared to B, C and D. A coincidence count is incremented when a loss is detected in channel A (master) followed by C and D but not B. Observation window length is 8 ADC samples.
- Channel B is compared to A, C and D. A coincidence count is incremented when a loss is detected in channel B (master) but not in A, C and D. Observation window length is 8 ADC samples.
- Channel C is compared to A, B and D. A coincidence count is incremented when a loss is detected in channel C (master) followed by A and B but not D. Observation window length is 8 ADC samples.
- Channel D is compared to A. A coincidence count is incremented when a loss is detected in channel D (master) followed by A within an observation window length 8 ADC samples. Channels B and C are irrelevant.

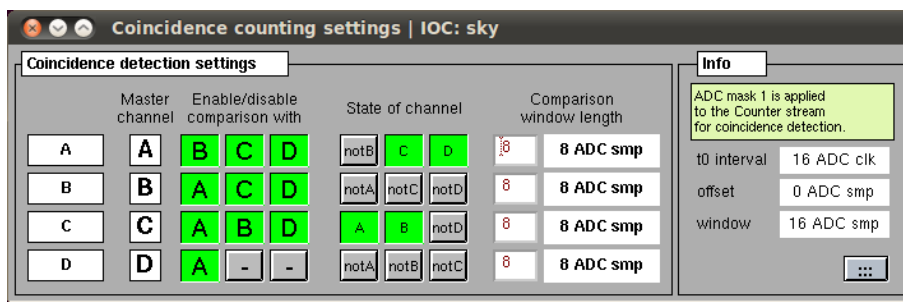


Figure 18: GUI screenshot for coincidence counting settings.

5.2.9 Read out limitations

SUM and AVG buffers are filled with highly decimated data rates especially if decimation factors are set to higher values. Buffers are written with data as it comes from the processing chain (=slowly). Due to software framework limitation, requesting the (SUM/AVG) data on trigger (with `-bEvent` switch and its corresponding mode in EPICS/TANGO), the readout request may fail.

Data is still being written correctly to the buffers and can be read out normally (without `-bEvent` and its corresponding mode in EPICS/TANGO). The write pointer will move to start position (0) at every acquisition trigger event.

In most extreme case (SUM decimation=4096, AVG decimation=262144), one AVG sample is written every 8.5 seconds (at 125 MHz sampling clock).

High trigger repetition may affect the SA data readout as well (with large decimation values). Take into account the decimation values, the sampling frequency and trigger repetition.

Avoid reading ADC buffers (raw, synthetic, integrated) in parallel with a higher repetition frequency (> 3 Hz). Due to CPU consumption, readout performance is degraded.

5.3 Input gain control

Each input channel contains a 31 dB variable attenuator. At full attenuation, the full-scale ADC range is ± 5 V (50 Ω input impedance) or ± 1 V (1 M Ω input impedance). Attenuation can be set for each channel individually with 1 dB setting resolution. Details of variable attenuators are given in Table 7.

Table 7: Input gain control.

Property	Value
Full attenuation	31.5 dB
Minimum attenuation	0.5 dB
Default attenuation (after reboot)	31.5 dB (full attenuation)
Setting step	1 dB

The following example shows how to read and set the attenuation for channel A:

```
root@libera:~# libera-ireg dump application.hk.attenuation.A
A=31
root@libera:~#
root@libera:~# libera-ireg application.hk.attenuation.A=20
```

Table 8 shows 5 selected input voltages with corresponding attenuation values to reach the ADC full scale. Tested with 1 MHz sine wave signal.

Table 8: ADC full scale vs attenuation.

ADC full scale		
Attenuation [dB]	1 M Ω input impedance	50 Ω input impedance
31	± 1.25 V	± 5.0 V
23	± 0.50 V	± 2.1 V
12	± 0.15 V	± 0.6 V
2	± 0.05 V	± 0.2 V
0	± 0.05 V	± 0.15 V

5.4 Input impedance selection

The input impedance can be set individually for each channel. It can be set to:

- 0: 1 M Ω ; for observing small and sporadic losses
- 1: 50 Ω ; for observing fast losses

When changing the input impedance, make sure the input signal is properly attenuated not to damage the input channel.

NOTE: Refer to Table 8 for maximum input amplitude allowance at selected input impedance.

The input impedance for channel A can be checked and set in the following way:

```
root@libera:~# libera-ireg dump application.hk.termination.A
A=0
root@libera:~#
root@libera:~# libera-ireg application.hk.termination.A=1
root@libera:~#
root@libera:~# libera-ireg dump application.hk.termination.A
A=1
root@libera:~#
```

5.5 Triggering options

The acquisition trigger event can be taken from three sources:

- Hardware input pulse (T2, raising edge)
- Software trigger (issued with a software command)
- Auto-triggering on ADC threshold (threshold is selectable per channel)

Acquisition trigger event starts processing on all four processing chains simultaneously.

To enable external trigger pulse (T2), set these parameters:

```
root@libera:~# libera-ireg application.signal_processing.trig_source=1
root@libera:~# libera-ireg application.triggers.t2.source=External
```

To send a software trigger (one pulse only), set these parameters:

```
root@libera:~# libera-ireg application.signal_processing.trig_source=1
root@libera:~# libera-ireg application.triggers.t2.source=Pulse
```

To enable auto-triggering, set this parameter (2 = Threshold):

```
root@libera:~# libera-ireg application.signal_processing.trig_source=2
```

The auto-triggering can react to a rising or falling edge. Selection is done in registry node:

```
libera-ireg application.signal_processing.threshold.autotrig.edge_detection
```

Auto-triggering can be set to:

- 0 ... falling edge (default)
- 1 ... raising edge

In case of auto-triggering, the first ADC sample (in any of the four signals) that exceeds the threshold starts the processing. After the auto-trigger event, the threshold detection is disabled. User has to re-enable the auto-trigger detection manually (set the trig_source to 0 or 1 and then back to 2).

For correct understanding of the »trigger« event, see Figure 19. The event stream (Chapter 5.8.1) records all events that happen at data processing. If acquisition trigger source is set to »External«, the data will be processed upon trigger T2 reception. Immediately after T2 reception (black color), the actual acquisition trigger is fired (red color). If acquisition trigger source is set to »Threshold«, the data will be processed as soon as the threshold value in one of the channels is exceeded. This trigger is shown as green color and followed by the acquisition trigger (red color).

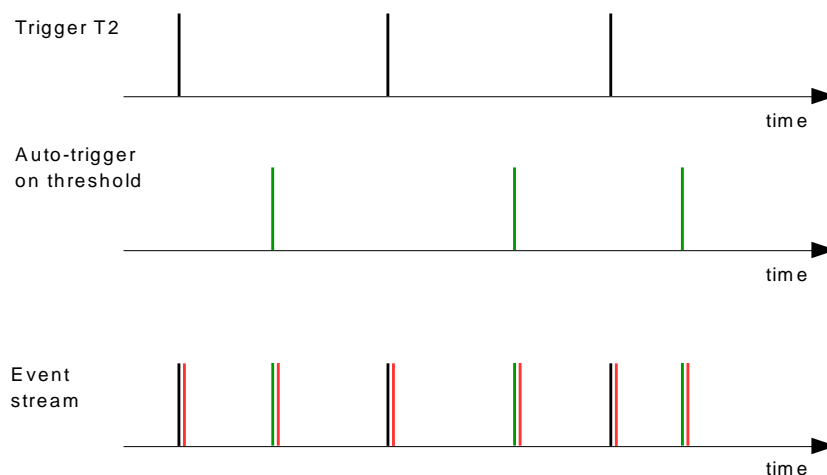


Figure 19: Description of "trigger" events.

5.6 Trigger delay

Trigger signal is used to start the data writing in to the RAM memory. Depending on the cable lengths and physical location of the unit, the trigger arrival to several Libera BLM units may be different. Libera BLM can delay the trigger signal. The setting resolution is in one LMT unit. One can issue the following command to set the trigger delay:

```
root@libera:~# libera-ireg application.triggers.t2.delay=0
```

LMT exact value can be calculated from oscillator's frequency which can be found in the libera-ireg tree.

$LMT = 1/ADC \text{ FREQUENCY}$

The trigger delay works as a real physical delay of the input trigger signal.

5.7 Channel line delay

To compensate a difference in cable lengths from the Libera BLM to detectors (connected to the same instrument), a configurable delay per channel is available to set. It is configurable with 1 ADC clock resolution (8 ns at 125 MHz) up to 32 ADC clocks. This is a fixed delay and works as a real delay line. The Trigger delay still works unchanged and is applied to all channels.

Setting is available in registry nodes:

```
root@libera:~# libera-ireg dump application.hk.channel_delay.
channel_delay
  A=0
  B=0
  C=0
  D=0
```

Default values are all 0. Values can be modified through a configuration file `/var/opt/libera/cfg/libera-blm.xml`

Note, that this file must be created before. To create the file, do:

```
root@libera:~# libera-ireg system.persistence.save{}  
system.persistence.save{}: done
```

Then, edit the values.

5.8 Sampling clock

Default Libera BLM sampling clock is set to 125 MHz. User can check it by issuing the command below.

```
root@libera:~# libera-ireg application.clock_info.adc_frequency  
application.clock_info.adc_frequency: 125000000
```

Depending on the input signal properties, user might calculate and set a new sampling clock frequency. The ADC sampling clock can be changed by editing the initialization script. Frequency can be set in the range between 80 MHz to 125 MHz. Note that the application daemon has to be stopped prior making any modification of the initialization script.

```
root@libera:~# /opt/etc/init.d/S50libera-blm stop  
Stopping libera-blm...  
Unloading Libera kernel module...  
root@libera:~#
```

Set the sampling clock frequency in the S50libera-blm file as shown below. In this example the sampling clock was set to 100 MHz.

```
root@libera:~# nano /opt/etc/init.d/S50libera-blm  
.   
.   
do_start()  
{  
    # Set ADC clock frequency  
    echo "Setting ADC clock frequency..."  
    # TODO: Obtain ADC frequency from a config file and write it to  
    #       /sys/bus/i2c/devices/0-0055/frequency.  
    echo 100000000 > /sys/bus/i2c/devices/0-0055/frequency
```

After saving the S50libera-blm file, start the application and verify the setting.

```
root@libera:~# /opt/etc/init.d/S50libera-blm start  
Setting ADC clock frequency...  
Configuring FPGA...  
Loading Libera kernel module...  
Starting libera-blm...  
root@libera:~#
```

5.8.1 T0 PLL

The ADC sampling clock can optionally be controlled by a software PLL. A reference signal is provided through a T0 connector on the instrument's back panel. The reference frequency is derived from the ADC sampling clock. Clock and decimation information is shown in:

```
libera-ireg dump application.clock_info

clock_info
  adc_frequency=125000000
  pll_ref
    decimation=125
    frequency=1000000
```

Description of parameters:

adc_frequency ... ADC sampling clock [Hz], set by the initialization script (see Chapter 5.7)

pll_ref.decimation ... division factor (integer) between the adc_frequency and T0 reference frequency

pll_ref.frequency ... T0 reference frequency to be applied at the T0 connector [Hz]

The 'pll_decimation' is configured in the /opt/libera/sbin/cfg/libera-blm.xml factory-default configuration file or in the /var/opt/libera/cfg/libera-blm.xml user-configuration file. See Chapter 4.4.2 for details how to handle configuration files. The pll_decimation can be set between 16 and 4096 (integer). Default is 125.

5.9 Event stream

Event stream is useful for detailed timestamp analysis and presence of different events. It provides the event's ID, its sequence number (count) and timestamp in LMT.

To read the event stream, see example (additional debug output is not shown in the example):

```
root@libera:~# libera-ireg signal application.signals.event -s5 -v
Signal data from node: cache.host-127-0-0-1:5679.app.application.signals.event
  id      count      timestamp      data
  3      1073804    27016886271757    0
  33      751      27017011153837    0
  3      1073805    27017011271757    0
  3      1073806    27017136271757    0
  3      1073807    27017261271757    0
```

Description of events is given in Table 9.

Table 9: Event ID description.

Event ID	Description
2	External trigger event
3	Counter stream event
4	Prescaled T0 event (for T0 PLL)
32	Auto-trigger event
33	Acquisition trigger event

5.10 BLD power supply and gain control

Libera BLM features four RJ-25 interfaces with power supply and gain control voltage pins for the BLDs. The power supply voltage and gain control voltage (its limit value) are configured through a dip switch panel on the instrument's back panel. See Chapter 7.3 for more details.

The software interface provides the information about the current power supply voltage and gain control voltage limit set values. The gain control voltage can be fine tuned within its limit value. See example output of the output values:

```
root@libera:~# libera-ireg dump application.detector
detector
  power_supply=5V
  vgc
    limit=1V
    output
      A=0
      B=0
      C=0
      D=0
```

The power supply voltage and gain control voltage limit values cannot be modified through software interface but only through a dip switch. To read currently set values, see example:

```
root@libera:~# libera-ireg application.detector.power_supply
application.detector.power_supply: 5V
root@libera:~# libera-ireg application.detector.vgc.limit
application.detector.vgc.limit: 1V
```

To read and set the gain control voltage, see example (for channel A):

```
root@libera:~# libera-ireg application.detector.vgc.output.A
application.detector.vgc.output.A: 0
root@libera:~# libera-ireg application.detector.vgc.output.A=0.7
root@libera:~# libera-ireg application.detector.vgc.output.A
application.detector.vgc.output.A: 0.699951171875
```

The actual gain control voltage value may slightly differ from the set one due to DAC resolution. The limit value depends on the gain control voltage limit value set by the dip switch. The value is given in Volts.

NOTE: Maximum output current (per channel) of the gain control voltage and power supply is limited to 30 mA. See Table 12 for more details.

NOTE: When the Vgc limit is changed in the DIP switch, the Vgc outputs go to 0 V. Voltage glitch of approximately 300 μ s is output just before the output goes to 0 V. Please set the output voltage to 0 before changing the DIP switch configuration to avoid the glitch.

5.11 Calibration functionality

Amplitude from the beam-loss detectors vary depending on at least three factors:

- Differences between the PMTs (tolerances, types, etc.)
- Gain control voltage setting of the PMTs
- Gain setting inside the Libera BLM

To calibrate the amplitude readouts, these three factors can be taken into account at data readout.

All output data (except the raw ADC) is multiplied with a factor that is calculated from the above factors:

$$A_{cal} = A_{raw} \times BLDCalib \times G \times AT$$

Where:

Acal	calibrated amplitude
Araw	raw amplitude (no correction)
BLDCalib	BLDCalib ... It is a calibration constant specific to each channel and the PMT.
G	It is a relative gain factor that depends on the setting of the gain control voltage.
AT	It corrects for the $10^{(Att/20)}$

Correction factor for the gain control voltage is specified for 8 different settings which are most commonly used. The actual gain control voltage that is set to the specific channel is compared to these 8 pre-defined settings. The closest value is then taken for the correction coefficient. See example:

User sets the gain control voltage (Vgc) to value **0.17 V**. Correction coefficient (G) is taken from the Vgc reference values (rounded to the closest value in table).

Vgc ref	0.00	0.30	0.40	0.50	0.60	0.70	0.80	0.90
G	NaN	334.5	33.25	4.97	1	0.26	0.0825	0.0313

In example case, the closest Vgc reference value is 0.30 V which gives the factor G=334.5.

Calibration can be enabled or disabled. When enabled, it is applied to these data paths:

- ADC Synth
- SUM
- AVG
- SA
- Postmortem

Calibration coefficients can be set independent to each channel.

Registry nodes that control the calibration functionality are shown below:

```
root@libera:~# libera-ireg dump application.calibration.
calibration
  enabled=false
  calculated_coeff
    A=1
    B=1
    C=1
    D=1
  bldCalib
    A=100
    B=100
    C=100
    D=100
  vgc
    A=[0,0,0,0,0,0,0,0]
    B=[0,0,0,0,0,0,0,0]
    C=[0,0,0,0,0,0,0,0]
    D=[0,0,0,0,0,0,0,0]
  g
    A=[0,0,0,0,0,0,0,0]
    B=[0,0,0,0,0,0,0,0]
    C=[0,0,0,0,0,0,0,0]
    D=[0,0,0,0,0,0,0,0]
```

Setting ranges:

bldCalib ... From 1 to 10000, integer values

vgc ... From 0 to 12, floating point values

g ... From 0 to 65535, integer values

5.12 Support for positive loss pulses

Beam loss detectors usually output a negative pulse that corresponds to the loss magnitude. All loss detection and processing logic in the Libera BLM is adapted to negative pulses. To support the beam loss detectors, that generate positive pulses, the ADC signal inside the Libera BLM can be inverted.

Once set, the functionality inverts the ADC data on all 4 channels.

Registry node that controls the ADC invert functionality is shown below

```
root@libera:~# libera-ireg application.hk.adc_invert
```

0 ... ADC data is normal

1 ... ADC data is inverted

6. Software

6.1 General overview

The Libera software is organized in a few layers, see Figure 20. In the bottom layer, there is Linux Kernel Module which communicates with FPGA. On top of that, there is application daemon (control algorithms and signal acquisition). The application communicates with outer world through the MCI API which uses registry and signal library underneath for accessing application properties, acquiring signals, changing the application behavior, etc. On top of MCI, adapters to various control systems can be implemented (EPICS, Tango, etc.). As a command line utility, the libera-ireg is implemented as a tool to easily explore the registry tree.

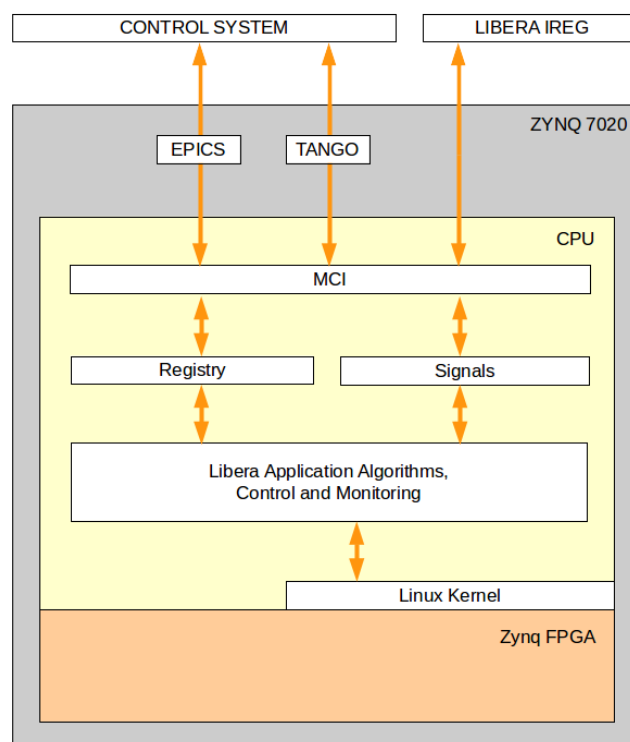


Figure 20: Libera BLM software structure

With such layered software, we hide low level details and we provide the user with a nice programming interface. At the same time, the API is not affected by internal upgrades of the software, providing stable environment, which enables very easy maintenance of the user's software above the MCI API. The eventual hardware changes (caused for example by the components obsolescence) are also masked by the MCI API.

6.2 Hardware components

The core of the instrument is the Xilinx Zynq, a dual core ARM CPU integrated with an FPGA and several communication controllers on the same chip. There are additional hardware components interconnected on the instrument supporting the functionalities, especially the four channel signal acquisition. The Xilinx documentation refers to the CPU as the processing system (PS) and to the FPGA as the programmable logic (PL).

The main function of the instrument is the signal acquisition of four channels and the storage of the raw and processed data to the RAM. The data can be retrieved over the Ethernet by the Control system. The SD card and the Serial Flash are there to support the startup (booting) of the instrument.

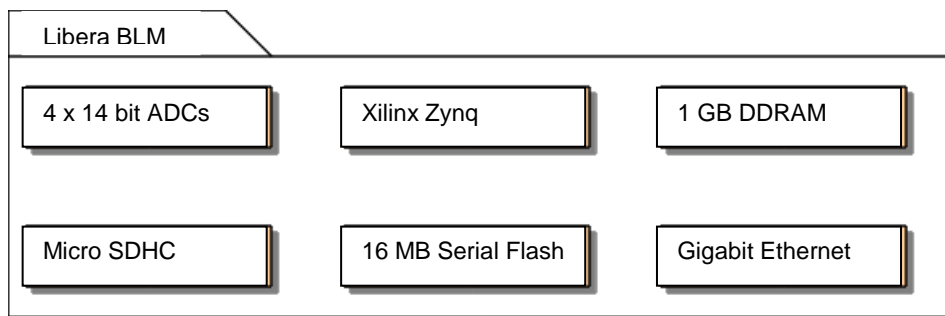


Figure 21: Hardware components.

6.3 Software components

At instrument's power ON, it first go through the boot sequence with the goal to startup Linux OS and the BLM application. When the system is up and running, it accepts connections via MCI on port 22 for SSH session(s). Several software modules are involved in these operations.

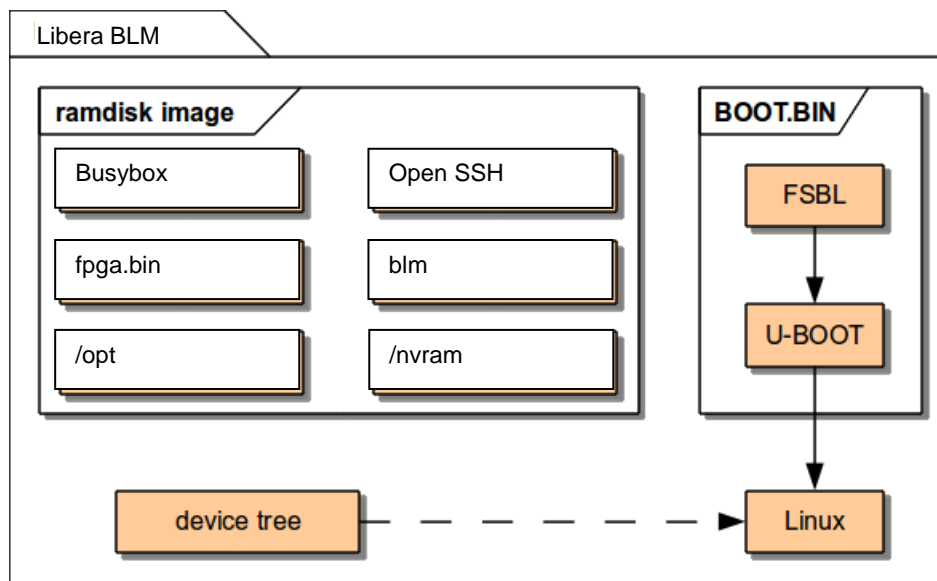


Figure 22: Software components.

Figure 22 shows the software components of Libera BLM:

- FSBL (First Stage Boot Loader) serves for initializing various Zynq components like DDR, UART, PLL clocks, communication controllers, etc.
- U-BOOT is a primary boot loader commonly used in embedded devices.
- BOOT.BIN is the container for FSBL and U-BOOT and can be loaded either from the SD card or from the Serial Flash.
- Linux is the kernel of the UNIX-like Operating System that provides services to the user space applications.
- The ram disk image contains the initial file system disk contents that are unpacked to the RAM and mounted there for the Linux Operating System to run. There are the configuration files and several other software modules that are automatically started.
- Busybox combines tiny versions of many common UNIX utilities into a single small executable.
- Open SSH server and client.
- blm is a daemon that initializes and configures the ADCs and the FPGA for the beam loss processing functionality.
- fpga.bin is the FPGA design file that is loaded
- /opt persistent storage
- /nvram partition for Libera BASE persistence (QSPI FLASH)

The BOOT.BIN with its components needs to be located on the instrument, either in the SD card or in the Serial Flash. The other components (Linux, device tree etc.) can also be stored on the instrument. These can be loaded over the network, e.g. from the TFTP server.

6.4 Booting

The boot process starts when the instrument is powered ON. ARM core is reset first and starts executing the boot ROM code.

6.4.1 Zynq boot ROM

The Zynq itself contains a small boot ROM that checks for the presence of the SD card. In case that it is inserted it loads the BOOT.BIN from the first FAT partition on that card. If there is no SD card, the BOOT.BIN is loaded from the QSPI Serial Flash.

6.4.2 FSBL

The BOOT.BIN is a container that can encapsulate several entries. The first entry must be the FSBL.elf ARM binary code. The execution is transferred to this component. Its purpose is to initialize and configures various hardware components, optionally load the FPGA design file and transfer execution to the next elf ARM binary file in the BOOT.BIN container.

6.4.3 U-BOOT

When the U-BOOT gets control, it initializes the UART for console input and output. It waits for 3 seconds before continuing the boot command. Within that time, the user can interrupt the boot sequence and enter in

interactive U-BOOT mode. Otherwise, the U-BOOT copies the Linux and device tree to the memory and transfers control to the Linux kernel.

6.4.4 Linux, device tree and RAM disk image

The kernel uses the device tree for the initialization of attached hardware devices. Also the kernel options (bootargs) setting is stored there and instructs the kernel to mount the root file system in RAM.

```
bootargs="console=ttyPS0,115200root=/dev/ram rw earlyprintk"
```

CMA is used for early reservation of the physically linear DDR memory for later utilization for DoD buffer(s) by the Libera BASE Linux kernel driver. The current CMA total reservation size is 480 MB, but the CMA pool is shared with other system DMA devices such as the 1000Base-T Ethernet and several others. They all consume very little amount of DMA memory, therefore at least 448 MB of CMA pool is available for DoD buffers.

The network settings are configured in the /etc/network/interfaces file. Default configuration waits the DHCP for 60 seconds. In case of timeout, it does fallback to static IP address 192.168.1.100.

The ramdisk image contains the initial file system with the busybox, OpenSSH, NTP daemon, and other BLM OS components. The BLM daemon is installed on /opt using the opkg package manager. The startup scripts are located in the /opt/etc/init.d.

6.4.5 Libera BLM daemon

The BLM daemon is started as the last startup service when the network is up and running. It configures the clock, and waits for the MCI connection.

NOTE: All buffers are empty upon start up and SA data stream does not output any data. For first buffer filling and SA data stream starting, one acquisition trigger is required.

6.5 Building

The environment used while preparing this instruction was the Ubuntu 12.04 64-bit system with several additional packages required for building the software. More details about those components can be found in Chapter 6.3. Separate build tasks put intermediate results to a stage directory. Finally, the ramdisk image build assembles all components together. Procedures for building FPGA designs and the FSBL are out of scope of this document and official Xilinx documentation should be consulted instead. Therefore, the fsbl.elf and fpga.bin files have to be present in the stage directory.

The Xilinx PlanAhead ISE or Vivado tools need to be installed for the arm cross compiler required to build the software. Instructions can be found in the Xilinx documentation. The WebPack license will be required and is free of charge.

There are several other packages required for building the software and should be installed on the build system.

```
sudo apt-get install git build-essential ia32-libs
```

Before building the software, the cross compiler environment needs to be set up so the tools are accessible in the system path:

```
$/usr/local/xilinx/14.5/ISE_DS/settings64.sh
$arm-xilinx-eabi-gcc --version
arm-xilinx-eabi-gcc (Sourcery Code Bench Lite 2012.09-105) 4.7.2
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

6.5.1 U-BOOT

The original U-BOOT source code was taken from the Xilinx github repositories <https://github.com/Xilinx/u-boot-xlnx> tagged with xilinx-v14.5.01 and locally modified on the i-tech branch. The modified repository is available on the accompanying DVD. It can be installed as bare repository and cloned to some local system or it can be converted to normal (non-bare) directly by copying it to the .git and checking out the i-tech branch as shown below.

```
$mkdir u-boot-xlnx
$cd u-boot-xlnx
$cp /media/cdrom/u-boot-xlnx.git .git
$git config --bool core.bare false
$git checkout i-tech
```

Please consult git documentation for more details on using and managing the git repositories. Issue the following commands in the u-boot-xlnx directory to build the u-boot binary from sources.

```
$export CROSS_COMPILE=arm-xilinx-eabi-
$make mrproper
$make zynq_libera_blm
$make all
```

After successful build, the u-boot binary is available and should be copied to the stage directory as u-boot.elf file. The differences made for Libera BLM from the Xilinx original can be examined with the following command:

```
$git diff xilinx-v14.5.01
```

6.5.2 Linux and device tree

Same approach as for the u-boot was used also for the Linux sources. The original Xilinx repository was cloned and i-tech branch was created on top of the xilinx-v14.5.01 tag for local changes. In order to build the Linux kernel from the modified sources, the repository can be cloned or copied in the same way as described for the u-boot sources.

```
$mkdir linux-xlnx
$cd linux-xlnx
$cp /media/cdrom/linux-xlnx.git .git
$git config --bool core.bare false
$git checkout i-tech
```

Commands for building the kernel are as follow below and should be run in the linux-xlnx directory. The mkimage utility is necessary for the last build step and should either be made accessible in the PATH or it can be installed with the uboot-mkimage package.

```
$make mrproper
$make ARCH=arm xilinx_zynq_defconfig
$make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- UIMAGE_LOADADDR=0x8000 uImage
```

After successful build, the kernel binary is available in arch/arm/boot/uImage and should be copied to the stage directory. Besides the kernel, there is also the device tree file included in the sources. It is called zynq-libera-blm.dts and can be found in the arch/arm/boot/dts directory among all other dts files. It should be compiled with the dtc utility and copied to the stage directory.

```
$scripts/dtc/dtc -I dts -O dtb -o $STAGE_DIR/devicetree.dtb arch/arm/boot/dts/zynq-
libera-blm.dts
```

6.5.3 boost library

Boost is required for the BLM application. Version 1.54 was used for testing the product and sources can be found on the accompanying DVD. It can be cross compiled for ARM with following commands:

```
$tar xf /media/cdrom/boost_1_54_0.tar.bz2
$cd boost_1_54_0
$./bootstrap.sh
$sed -ie "s/ using gcc ;/ using gcc : arm : arm-xilinx-linux-gnueabi-gcc ; /" project-
config.jam
$./b2
```

6.5.4 Libera BLM daemon

The Libera BLM daemon is built with the Makefile that is part of sources.

6.5.5 Boot image

The boot image build gathers the results of all dependent components described above from a common stage directory. It produces the SD card image file that can be transferred to the SD card with the dd command. It also produces a tar archive that contains the component to be copied on the tftp server for the network boot.

7. Specifications

7.1 General specifications

General specifications are gathered in Table 10.

Table 10: General specifications

	Specification
Power supply	Power over Ethernet, 48 V; IEEE 802.3af
Power consumption	Max 15 W
Operating environment	Temperature: 10 – 35° C / 50° – 95° F Relative humidity: 20 – 80 %
Warm up time	2 hours
Pollution degree	2
Installation category	II
A/D converter	2x dual-channel, 14-bit granularity, DC coupled
Sampling rate	80 MHz to 125 MHz
Processing engine	Xilinx ZYNQ 7020, dual ARM Cortex-A9 CPU cores
Memory	DDR3 SDRAM 1 GB, 32-bit, 533 MHz Removable MicroSD (for FPGA binary, OS image)
Measurement bandwidth @ 50 Ω input termination	Large signal bandwidth ~ 35 MHz Small signal bandwidth ~ 50 MHz

7.2 Hardware inputs

The input analogue channels (A, B, C, D) are designed to work with signals up to a maximum* of:

- ± 1 V at 1 M Ω input termination
- ± 5 V at 50 Ω input termination

* Check Table 8 for details

Table 11. Hardware inputs specifications.

Input connector	Type	Description
A, B, C, D	SMA-F	Selectable input termination (50 Ω , 1 M Ω) Maximum amplitudes: <ul style="list-style-type: none"> • ± 5 V CW at 50 Ω • ± 1.25 V CW at 1 MΩ

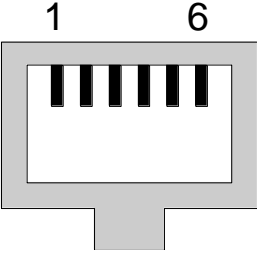
Input connector	Type	Description
T0, T1, T2	EPL.00.250.NTN	LVTTTL (3.3 V), ~10 k Ω termination Raising edge detection 40 ns minimum pulse width Maximum trigger repetition: <ul style="list-style-type: none"> T0: 0.1 MHz – 5 MHz (typical) T1: 1 Hz T2: 10 Hz

CAUTION: The input amplitude must not exceed the allowed value as it may cause irreversible damage to hardware components. These damages are not covered by warranty.

7.3 Hardware outputs


Libera BLM contains 4x RJ-25 6P6C connectors that provide power and gain control voltage to beam loss detector with a photosensor. Power supply voltage and gain control voltage limitation is common to all 4 connectors. Gain control voltage can be set to each channel individually. Pin description of the RJ-25 connector is given in Table 12.

Table 12: BLD connector pin description.

	pin		Description	Maximum consumption	Maximum power dissipation
	1	+Vout	(+) power supply voltage	30 mA	~ 300 mW per 2 channels $P=(V_{out}-V_{gc})*I_{vgc}$
	2	GND	Ground	/	
	3	-Vout	(-) power supply voltage	30 mA	
	4	GND	Ground	/	
	5	GND	Ground	/	
	6	Vgc	Gain control voltage	30 mA	

The power supply voltage is selectable between ± 5 V and ± 15 V and is fixed (not software configurable). Gain control voltage is adjustable by software within 0 and +Vout range. For safety reasons, the maximum gain control voltage can be limited by a dip switch. Gain control voltage is given with 12-bit granularity. Description of dip switch combinations is given in Table 13.

Table 13. Dip switch configuration.



		Gain control voltage limit			
switch		1 V	2 V	5 V	12 V
1		off	on	off	on
2		off	off	on	on
		Power supply voltage			
switch		± 5 V	± 10 V	± 12 V	± 15 V
3		off	on	off	on
4		off	off	on	on

The power supply and gain control voltage circuit is shown in Figure 23.

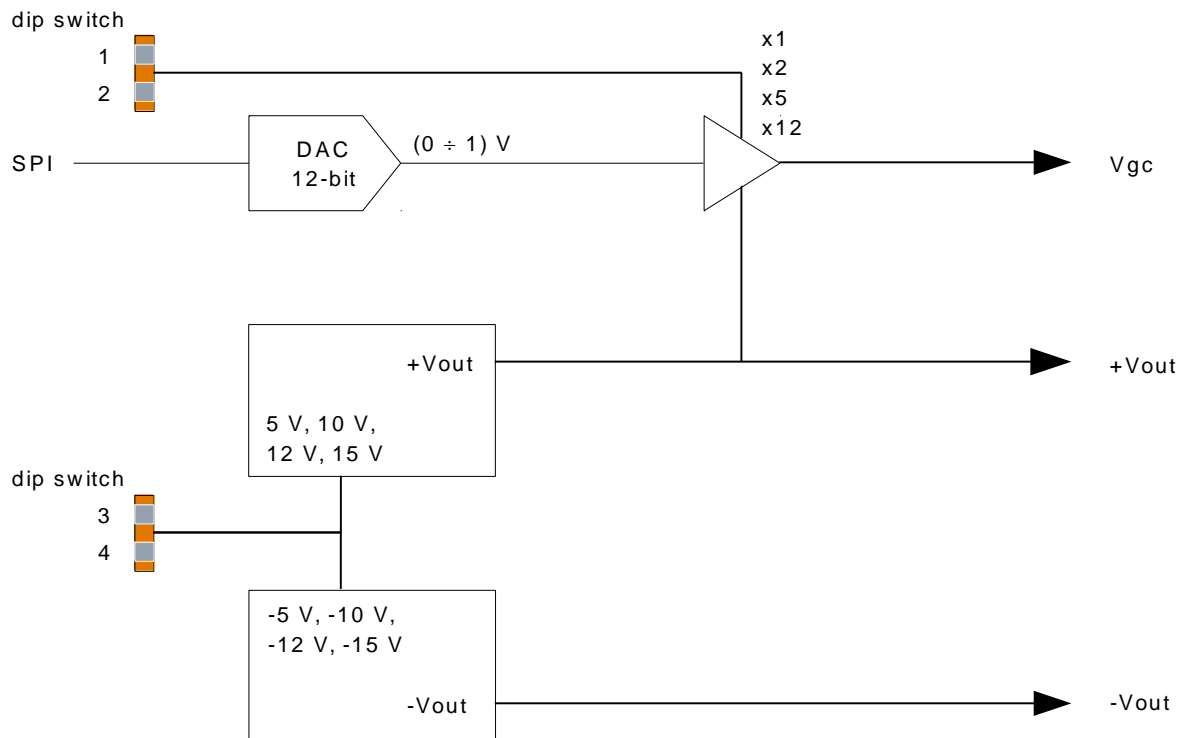


Figure 23: Hardware output circuit.

7.4 Enclosure

Instrument's dimensions are as follows (Figure 24):

- Width: 210 mm / ~8.27 in
- Depth: 210 mm / ~8.27 in
- Height: 44 mm / ~1.73 in
- Weight: 1.4 kg / ~ 3.1 lbs



Figure 24: Instrument's dimensions

8. Appendix A: Beam Loss Detector

Beam Loss Detector is optionally delivered with the Libera BLM. It consists of a γ -scintillator rod and the photo-multiplier. It can be covered with the lead cover that (partially) prevents the scintillator from the X-rays.

8.1 Specifications

Summary of specifications are given in Table 14.

Table 14: Specifications for the beam loss detector.

Part	Item	Specification
Beam Loss Detector	Dimensions (height x width x depth)	Approximately 220 x 25 x 25 (without the fitting holder)
	Weight	Approximately 150 g (without the lead cover)
	Operating temperature	+10°C to +40°C
Scintillator rod	Model	Scionix, EJ-200
	Length	100 mm
	Diameter	22 mm
	Cover	Metalized cover
Photosensor module	Model	Hamamatsu 10721-110*
	Power supply voltage	+4.5 V to +5.5 V
	Gain control voltage	< 1.1 V (input impedance 1M Ω)
	Rise time	0.57 ns
	Dark current	1 nA
	Peak sensitivity wavelength	400 nm

* subject to change (10721-110 or 10721-210)

8.2 Hardware interfaces

Beam loss detector contains 2 interfaces:

- SMA: signal output
- RJ-25: power supply and gain control voltage

RJ-25 pinout is given in the label on the detector's housing. Refer also to Table 12 for more information. Note that some photosensor modules do not require -Vcc voltage. Figure 25 shows the interfaces on the beam loss detector.



Figure 25: Interfaces on the beam loss detector.

8.3 Installation

There are no special limitations on beam loss detector physical installation. The fitting holder is provided with 2 holes for fitting onto the solid surface. The fitting holder can be mounted on the aluminum housing in 2 directions.

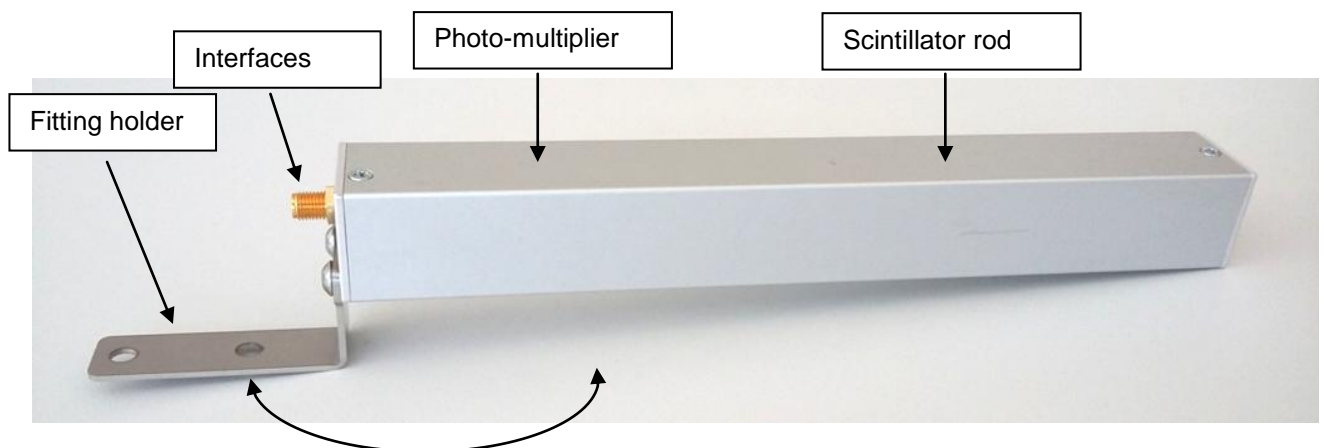


Figure 26: Beam loss detector.

More at www.i-tech.si

Visit our website to read more about Libera products, download conference papers on the use of Libera at different accelerators around the world, subscribe to the I-Tech Newsletter and learn about the next gathering of the community at the Libera Workshop.

Technical support

Prompt and reliable. You can ask for on-site support or we can assist you remotely. You are also welcome to join us at the Libera Workshop training sessions to get the most out of Libera products.

