GSI Data Acquisition System MBS
Release Notes V6.3

Jörn Adamczewski-Musch, Nikolaus Kurz, Sergei Linev

November 10, 2017

GSI, Gesellschaft für Schwerionenforschung mbH
Planckstraße 1, D-64291 Darmstadt
Germany
Tel. (0 6159) 71–0
Contents

1 MBS Release V6.3
   1.1 General Remarks .................................................. 3
   1.2 New Features in MBS Version 6.3 ................................. 4
   1.3 Support for 64bit Linux on PC Hardware .......................... 5
   1.4 Linux device driver software mbspex for PCIe readout systems with KINPEX .............................. 5
       1.4.1 Kernel module and library .................................... 6
       1.4.2 Changes in user readout function ............................. 6
       1.4.3 Command line tool gosipcmd ................................. 8
       1.4.4 Front-end control GUIs ....................................... 9
       1.4.5 New readout pipe type 4 .................................... 11
   1.5 Remote control of MBS with DABC ................................ 11
       1.5.1 New control sockets and HTTP server ....................... 11
       1.5.2 Web browser GUI ............................................ 14
   1.6 White Rabbit timing receivers .................................... 15
       1.6.1 Software installations ...................................... 15
       1.6.2 Device drivers .............................................. 15
       1.6.3 User readout function ...................................... 16
       1.6.4 Direct TLU register access ................................ 17
   1.7 Mass storage interfaces ......................................... 18
       1.7.1 RFIO disk server ............................................. 18
          On the RFIO Linux node: ..................................... 19
          On the mbs: ..................................................... 20
       1.7.2 Legacy RFIO tape server ..................................... 20
       1.7.3 RFIO gateway to new GSI storage interface LTSM .......... 21
          On the RFIO-LTSM Server node: .............................. 21
          On the mbs: ..................................................... 23
   1.8 New GSI boot servers with dedicated VLAN ...................... 23
   1.9 EPICS installations .............................................. 24
   1.10 Miscellaneous ................................................... 24
List of Figures ....................................................... 25
Bibliography .................................................................. 27
Chapter 1

MBS Release V6.3

MC_MBS_REL_63

1.1 General Remarks

These release notes can be retrieved as pdf file from the web page:
https://www.gsi.de/mbs

They cover all changes for MBS version 6.2 to 6.3. Please use these release notes in conjunction with the MBS user manual and the previous release notes v4.3, v5.0, v5.1, and v6.2 also available from the MBS homepage stated above.

Since November 1, 2017, the MBS version 6.3 has become the new production version. MBS version 6.3 replaces the old production version 6.2. The new MBS and its commands can be used as before. To access and use the new production version a few actions have to be taken:

1. Make sure that in the .login script in the home directory of your Linux (or LynxOS) account the command source /mbs/prodlogin.com is present.


3. If you had to change the .login script, logout and login again to make the changes active.

   In any case you can issue the command mbslogin prod to set the production version and mbsversion to check which version you are working with from the MBS node prompt.

4. All user functions and programs (m_read_meb, m_collector) have to be recompiled and linked! Your current make files are most probably not valid anymore. A new generic Makefile can be retrieved from /mbs/v63/exa/v63. If you have no other dependencies as f_user.c you can directly use this Makefile. Otherwise you need to modify it in the
usual way. For both cases a make clean followed by make in your MBS working directory is sufficient to rebuild your programs.

In case of troubles don’t hesitate to contact one of the following MBS maintenance developers:

- N.Kurz, GSI Tel.: 2979, E-Mail: N.Kurz@gsi.de
- J.Adamczewski-Musch, GSI Tel.: 1337, E-Mail: J.Adamczewski@gsi.de

1.2 New Features in MBS Version 6.3

The new MBS version 6.3 provides several new features, which will be described in further sections.

1. Support of 64bit Linux for X86 PCs
2. Linux device driver software mbspex and commandline tool gosipcmd (alias 'goc') for KIN-PEX based MBS readout systems
3. Several dedicated Qt control GUIs for mbspex front-ends POLAND, FEBEX, NYXOR, APFEL
4. Web browser remote control GUI via DABC HTTP server with improved MBS communication sockets
5. Support of White Rabbit timing receivers on RIO4/IPV VME (VETAR2A) and X86 PCI Express (PEXARIA, EXPLODER5A)
6. Support of new GSI storage interface LTSIM, local RFIO is still maintained
7. New MBS boot- and file servers at GSI with dedicated VLAN
8. Installation of EPICS control system on MBS Linux platforms
9. Miscellaneous
**MBS** version 6.3 supports the following processor platforms and operating systems:

- LynxOS 2.5: CVC, E7, RIO2, PC
- LynxOS 3.1: RIO3
- LynxOS 4.0: RIO4, PC
- Linux Debian Jessie 3.2, 3.2_64: PC
- Linux Debian Stretch 4.9_64: PC
- Linux Sugarhat 2.6.33: RIO4
- Linux DENX 3.3: IPV

Please note that older PC Linux OS Debian 5 and 6 (kernel 2.6) are not provided anymore by default with **MBS** v6.3! Additionally, the older RIO4 Linux OS Sugarhat (kernel 2.6.24) is not supported anymore with **MBS** v6.3. If you need an installation of v6.3 on these platforms, please contact the **MBS** maintenance developers.

### 1.3 Support for 64bit Linux on PC Hardware

The complete source code of the **MBS** framework has been remanufactured to work also on 64 bit Linux systems. This covers all data pointer types, the core data structures and the handling of the setup tables [5]. This has been tested on several X86 PC nodes running with Debian 7 (kernel 3.2.0-4-amd64). As a benefit from the enhanced address space, a larger pipe memory can be set up on such systems.

### 1.4 Linux device driver software *mbspex* for PCIe read-out systems with KINPEX

The driver software for the PCIe optical receiver boards PEXOR and KINPEX has been newly implemented for x86 Linux platforms [2] [5]. This consists in a new kernel module *mbspex* with corresponding C library. Additionally, a command line tool *gosipcmd* allows front-end configuration and controls from the system shell. As both kinds of hardware are handled by the same software, in the following the newer name KINPEX is used for either of the boards.

The proprietary **GSI Optical Serial Interface Protocol (GOSIP)** has been encapsulated into this driver software. So the communication between front-ends and readout host PC can be handled transparently with high-level C functions. Moreover, these functions are safe against concurrent access since they will lock critical sections at the kernel module level. If the MBS user readout function calls *mbspex* library functions only, a safe read-out will be possible concurrently with external control system access like *gosipcmd*. Backward compatibility to existing readout code is provided by the new driver.

Most details of this software have been presented on IEEE RealTime conference 2016 and will be available as publication in IEEE TNS [7].
1.4.1 Kernel module and library

The mbspx.ko kernel module [3] implements all functionalities required to communicate with the KINPEX board and the connected front-ends. The main features are:

1. `ioctl()` functions for basic gosip protocol functionalities, like initialization of chains, front-end field bus access, and token data requests
2. `ioctl()` functions for DMA operations to send data from PEXOR buffers to any destination pointer in the MBS pipe.
3. `ioctl()` functions for intrinsic token data requests from all connected front-end slaves, both in sequential (“direct DMA”) and parallel readout mode
4. concurrent access to the PEXOR device, because all `ioctl()` calls are protected by a kernel mutual exclusive semaphore.
5. broadcast I/O to all initialized devices at the SFP chains
6. configuration of several front-end registers in a single `ioctl()`
7. backward compatibility to legacy readout code (`mmap()` of pexor control registers) without concurrency protection
8. handling of `mmap()` to map the pre-boot reserved pipe memory to user space
9. register and handle any user space memory as scatter-gather DMA list for new pipe type 4
10. export some control registers of KINPEX and TRIXOR as Linux sys filesystem handle at `/sys/class/mbspex/pexor0`

The user space library libmbspex.so [4] offers C functions to transparently work with these mbspex device driver possibilities. For the GSI MBS installation, this library and the corresponding include headers are provided at the `/mbs/driv/mbspex` mount point for each X86 PC architecture. The user readout function `f_user.c` may use such functions and link to the correct library by means of the appropriate installation path taken from pre-defined environment variables.

1.4.2 Changes in user readout function

In order to achieve concurrent control system access to the front-end control registers during the MBS data acquisition running, the user readout function `f_user.c` has to call the “critical section protected” functions of libmbspex.so only.

Many common readout code examples will enable this mbspex mode with a compile-time definition `#define USE_MBSPEX_LIB 1`. By commenting this line at the top of the source file (and recompiling) it is possible to switch back to the legacy code with control of the GOSIP
functionality solely by user space access to the KINPEX registers. This can be useful to check readout functionality when migrating to the new driver software. **For concurrency protected DAQ it is strongly recommended to enable the mbspex mode!**

Following mbspex calls are necessary in different parts of the readout code:

**f_user_get_virt_ptr()** Call at first `int handle=mbspex_open()` to acquire the file descriptor to the mbspec character driver. This handle is used in all subsequent calls.

**f_user_init()** Here different functions are useful:

- Call `mbspex_slave_init(int handle, long lsfp, long n_slaves)` once for each chain to initialize the SFP chain `lsfp` with the number of connected front-ends `n_slaves`
- Use `mbspex_slave_wr(int handle, long lsfp, long slave, long slave_off, long l_dat)` to set values `l_dat` to any address `lslave_off` on the front-end number `lslave connected to SFP lsfp`
- Use `mbspex_slave_rd(int handle, long lsfp, long slave, long slave_off, long * l_dat)` to read values into local memory `* l_dat` from any address `lslave_off` on the front-end number `lslave connected to SFP lsfp`

**f_user_readout** Functions to request and retrieve data:

- `mbspex_send_and_receive_tok(...)` for sequential request and receive data from each SFP chain: - The data will be transferred with "direct DMA" to the destination pointer in pipe memory. **This call is protected against concurrent GOSIP slow control access.**
- `mbspex_send_and_receive_parallel_tok(...)` for parallel request and receive data from all configured SFP chains: - The data will be transferred from front-ends to the memory on KINPEX boards and then via DMA to the destination pointer in pipe memory. Alignment and MBS padding words are added in the kernel module. **This call is protected against concurrent GOSIP slow control access.**
- Other basic token request and DMA operations (not concurrency safe!):
  - `mbspex_send_tok(...)` - request data from one ore many SFP chains
  - `mbspex_receive_tok(...)` - wait for gosip token received
  - `mbspex_get_tok_memsize(...)` - find out size of received SFP data in KINPEX buffer
  - `mbspex_dma_rd(...)` - start DMA read from KINPEX buffer to host memory (pre-boot reserved memory pipe, type 3)
  - `mbspex_dma_rd_virt(...)` - start DMA read from KINPEX buffer to host memory (virtual memory pipe, type 4, see 1.4.5)
1.4.3 Command line tool **gosipcmd**

The command line tool *gosipcmd* (alias "goc") works as a shell application on top of *libmspex* [2][7].

It provides interactive access to the KINPEX board and to the registers of the front-end cards connected via gosip protocol. Any front-end card is addressed by the index of the SFP chain (0,...,3) and the position of the card in the chain (0,...,255). The returned read values are passed to stdout and can thus also be consumed by a higher level application like a controlling GUI (graphical user interface) or a web server.

The *gosipcmd* functionalities cover:

- reset KINPEX board, initialize SFP chains
- read/write any address on front-end slave
- incremental read/write from start address
- register bit manipulation
- broadcast mode: read/write same register at all connected front-ends
- configure/verify with script files *.gos*
- plain or verbose, hexadecimal or decimal output mode

A more complete list of available options can be printed using *gosipcmd -h*:

```bash
X86L-59 mbs > gosipcmd -h
***************************************************************************
gosipcmd for mbspex library
v0.4242 4-Nov-2016 by JAN (j.adamczewski@gsi.de)
***************************************************************************
usage: gosipcmd [-h|-z] [-i|-r|-w|-s|-u] [-b] | [-c|-v FILE] [-n DEVICE |-d|-x] [sfp slave [address [value [words] | [words]]]]
Options:
  -h : display this help
  -z : reset (zero) pexor/kinpex board
  -i : initialize sfp chain
  -r : read from register
  -w : write to register
  -s : set bits of given mask in register
  -u : unset bits of given mask in register
  -b : broadcast io operations to all slaves in range (0-sfp)(0-slave)
  -c FILE : configure registers with values from FILE.gos
  -v FILE : verify register contents (compare with FILE.gos)
  -n DEVICE : specify device number N (/dev/pexorN, default:0)
  -d : debug mode (verbose output)
  -x : numbers in hex format (defaults: decimal, or defined by prefix 0x)
Arguments:
  sfp - sfp chain -1 to broadcast all registered chains
  slave - slave id at chain, or total number of slaves. -1 for internal broadcast
  address - register on slave
  value - value to write on slave
  words - number of words to read/write/set incrementally
Examples:
  gosipcmd -z -n 1 : master gosip reset of board /dev/pexor1
  gosipcmd -i 0 24 : initialize chain at sfp 0 with 24 slave devices
  gosipcmd -r -x 1 0 0x1000 : read from sfp 0, slave 0, address 0x1000 and printout value
  gosipcmd -r -x 0 3 0x1000 5 : read from sfp 0, slave 3, address 0x1000 next 5 words
  gosipcmd -r -x 1 3 0x1000 10 : broadcast read from sfp (0..1), slave (0..3), address 0x1000 next 10 words
  gosipcmd -r -x 1 -1 0x1000 10 : broadcast read from address 0x1000, next 10 words from all registered slaves
  gosipcmd -w -x 1 0 0x2000 AB FF : write value 0x2A to sfp 0, slave 0, to addresses 0x2000-0x200FF
  gosipcmd -w -x 0 20000 30004c 1 : broadcast write value 1 to address 0x20004c on sfp (0..1) slaves (0..3)
```
The tool *gosipcmd* is provided for all **MBS** X86 Linux installations (hosts X86L-nn).

### 1.4.4 Front-end control GUIs

Several Graphical User Interfaces (GUI) applications have been developed to monitor and control the properties of different kinds of front-end boards at the **GOSIP** read-out chain. They are based on the Qt graphical library and are executed locally on the **MBS** readout nodes hosting the KINPEX board. Communication between such GUIs and the read-out slaves can be implemented either via shell execution of *gosipcmd*, or by calling the functions of *libmbspex* directly in the GUI.
Whereas the first approach can decouple the GUI from the actual device driver software, the second approach is better in terms of call latency. Therefore the GOSIP control GUIs deployed at the MBS installations are all directly linked to libmbspex.

The GosipGui base class Qt widget implements the common functionalities for all GOSIP front-ends, such as:

- general gosip communication interface
- resetting the KINPEX board
- initializing the SFP chains, selection of the current front-end
- managing multiple slave configurations with broadcast feature
- provide file interface to save and load configuration of all slaves
- buttons with virtual callback interfaces for front-end initialization, register set-up, status show, and data dump
- command output and debug terminal, status display

A number of front-end GUIs have been implemented as sub-classes of GosipGui. They provide the special control features of each hardware kind as widgets within an embedded scroll area, while the GosipGui frame keeps the same GUI elements for all implementations.

The following GUIs are installed at MBS Linux nodes and can be started after mbslogin (see 1.1) by the given alias names:

poland Starts PolandGUI for the POLAND charge frequency converter system.

febex Starts FebexGUI for FEBEX3 12/14 bit pipelining ADC systems. Optionally, the TUM-addon baseline adjustment is provided

nyxor Starts NyxorGUI to control NYXOR or GEMEX front-ends equipped with the n-XYTER chip.

apfel Starts ApfelGUI for APFEL readout system configuration, control, and tests. Implements a benchmarking suite with automatic sequencer for initial characterisation of the boards at deployment.

As an Example, Fig. 1.1 shows the GUI for the FEBEX3 front-end board.
1.4.5 New readout pipe type 4

A new readout pipe type 4 has been introduced to the MBS framework. For x86 Linux platforms this allows to allocate regular user space memory as readout buffer, instead of using coherent memory outside the kernel region reserved at boot time. Especially on 64 bit architectures this promises a larger pipe with more flexible set up. The DMA read out into such scatter gather memory pipe has been implemented for PEXOR-type boards in the mbspex device driver. The scatter-gather DMA functionality has to be treated by the kernel module, since the DMA engine of the KINPEX board’s FPGA code is not yet capable to treat scatter-gather memory. A read out test of POLAND/QFW front-ends proved that pipe type 4 was working in principle. Due to fragmentation of the user memory, the performance of such scatter gather DMA was about 5% less than with the coherent pipe type 3, and showed larger event rate fluctuations in time. Only when MBS was started directly after rebooting the machine, pipe type 4 could achieve about the same performance as type 3, because of minimum initial user space page fragmentation [5].

1.5 Remote control of MBS with DABC

1.5.1 New control sockets and HTTP server

Remote control of MBS via tcp/ip sockets is for a long time possible: the status server can deliver run state and rate information to monitoring clients like Go4, and multiple MBS nodes can receive commands via the prompter socket.

For MBS v6.3 two new socket channels have been introduced, mostly intended for control with the C++ software framework Data Acquisition Backbone Core (DABC) [6]. These channels may be started by new MBS commands:

- a remote logging server logrem can send log output to a control client. It can be started from the dispatcher process with the commmand start logrem.

- a remote command channel cmdrem in the dispatcher process can insert additional commands beside the local MBS console input. It can be started with the command start logrem.

When the MBS dispatcher console is invoked with argument mbs -dabc, these control sockets are started already at the beginning and wait for remote clients to be connected:

X86L-59: adamczew > mbs -dabc
-X86L-59:msg_log :Message logger running
-X86L-59:dispatch :- starting dabc remote logger socket...
mbs> -X86L-59:dispatch :- starting dabc remote command socket...
-X86L-59:msg_log :logrem: create listener for port 6007
-X86L-59:dispatch :cmdrem: create listener for port 6019

The DABC framework has different possibilities to use these channels:
For remote console or batch mode, the executable `mbscmd` can control the MBS dispatcher process via the `cmdrem` socket. Additionally, for older MBS versions ≤ v.6.2, `mbscmd` can also steer the prompter process via its already provided multi-node control socket. This can be useful if the state of the DAQ (start, stop, filename) shall be changed from another application, e.g. a slow control sequencer. The MBS data acquisition may even be steered from within the online monitoring program Go4\(^1\) by executing `mbscmd`, depending on particular analysis conditions.

For remote graphical user interface, DABC provides a HTTP server application that allows to control MBS from a regular web browser:

- The DABC web server application forwards HTTP command requests via the `cmdrem` channel to the dispatcher.

\(^1\)http://www.gsi.de/go4
– it retrieves log messages from the logrem channel,
– it can request the DAQ status from the MBS status socket,
– it can display this information to a web browser and allows interactive control

Figure 1.2 illustrates these features: The MBS dispatcher process can open 3 control sockets (cmd, log, status). A specific DABC plug-in provides a C++ API to use them. So a separate DABC process can connect to such channels and provide an HTTP server with a JavaScript control GUI for MBS in any web browser. Additionally, the tool mbscmd can be used to send MBS commands from any shell to the MBS process. This commandline tool may be also used from a Go4 online analysis to automatically switch parameters of the DAQ. Finally, as Go4 analysis may also provide a HTTP server, everything can be fully controlled from web browser tabs.

Because DABC has been installed on all MBS dedicated PC Linux systems at NFS mount-point /dabc, these remote control tools are available on all MBS PC Linux nodes. After setting the environment by invoking alias command dabcllogin, the mbscmd tool is found in the command PATH.

To start the DABC web server process, the following alias commands are predefined (after mbslogin, see 1.1):

webmbs < httpport > : Starts the DABC web server with port number < httpport > and connects to MBS dispatcher running on the local node. The MBS has to be started first in another shell with mbs -dabc

webmbsrem < mbsnode > < httpport > : Starts the DABC web server with port number httpport and connects to MBS dispatcher running on the host mbsnode. This allows to control MBS systems with no local DABC installation, e.g. LynxOS or PowerPC Linux systems. The MBS has to be started first on the node mbsnode with mbs -dabc.

webprm < httpport > : Starts the DABC web server with port number httpport and connects to MBS prompter running on the local node. The MBS has to be started first in another shell with prm -r localhost. This is a method to control older MBS version 6.2 where the newly control sockets are not yet implemented. This is not recommended for MBS v6.3!

webprmrem < mbsnode > < httpport > : Starts the DABC web server with port number httpport and connects to MBS prompter running on the host mbsnode. This allows to control MBS systems with no local DABC installation, e.g. LynxOS or PowerPC Linux systems. The MBS has to be started first on the node mbsnode with prm -r <dabcnode>, with dabcnode being the hostname of the machine where DABC webservice is running. This is a method to control older MBS version 6.2 where the newly control sockets are not yet implemented. This is not recommended for MBS v6.3!
1.5.2 Web browser GUI

Once the webserver is started, it can be accessed from any web browser with the address http://dabcnode.gsi.de:httpport. At startup, the default DABC tree display of all registered web objects [6] is presented in the browser.

In addition to this, a dedicated MBS control GUI has been developed for this DABC web server using JavaScript with the jQuery UI environment. It can be started by clicking the icon MBS/localhost/ControlGUI in the default DABC object tree hierarchy view.

Figure 1.3 shows a screenshot of such a GUI. The dispatcher interactive command line appears in the browser and offers additional shortcut buttons for mostly used commands, like start/stop acquisition, @startup, @shutdown. Data taking commands and status is shown in a separate tile with control buttons for open/close file, file mode and RFIO/local disk set up. The run or file status of the acquisition is visualized by red or green background colours of the control tiles. Data rates are displayed both as graphical gauge or trending rate meters, and as text terminal output of the previously known rate, rash, rast, or ratf shell commands of MBS. Automatic refresh frequency of the browser display can be set in a dedicated box. Moreover, when running on MBS Linux PC, the DABC web server also offers to call the front-end configuration tool gosipcmd (see 1.4.3), so a command line and output box for this tool can optionally be shown in the browser. Additionally, a dedicated front-end configuration GUI, like POLAND setup, may be started by a button in a different browser window. As an example, a web version of the POLAND Qt control
GUI has been implemented for the web server.

1.6 White Rabbit timing receivers

The future FAIR general machine timing distribution will be based on the White Rabbit system. Since 2014 it is possible in MBS to latch and read out such White Rabbit timestamp at the moment of the DAQ trigger, and therefore to synchronize independently running systems [1].

For such features the White Rabbit timing receiver (WRT) hardware PEXARIA (PCI Express), EXPLODER5A (PCIExpress, USB), and VETAR2A (VME) is supported by MBS for Linux systems. These WRTs provide a special "Time Latch Unit" (TLU) that can record the time stamp when an input signal changes, e.g. by the trigger signal.

1.6.1 Software installations

The White Rabbit drivers and user software from the most recent official GSI-CSCO releases Balloon and Cherry are installed for all MBS Linux platforms at the locations:

- /mbs-driv/white_rabbit/balloon
- /mbs-driv/white_rabbit/cherry

They have been compiled directly from the respective git repository checkout. For VME based RIO4 Sugarhat Linux, special patches and build scripts had to be developed to cope with the fact that such Linux is too old for the standard procedures of the release, and partially it misses some of the required tools.

After mbslogin (see section 1.1) the command PATH and the environment is set automatically such that the correct executables and libraries for the current system are used. Apart from the PCIe and USB device drivers, mostly the etherbone tools (eb-ls, eb-info, eb-flash, etc.) are provided. Additionally, the libetherbone library and its appropriate includes can be used from the MBS readout function. The slow control SAFT library is not necessarily required by MBS, but may also be deployed for these nodes in the future.

1.6.2 Device drivers

For PCIe systems (PEXARIA and EXPLODER5A), the device drivers of the "official" software release from the GSI accelerator timing group (CSCO) are used without modifications. These kernel modules are installed under the locations described in section 1.6.1 in the appropriate subdirectory PLATFORM_OS_VERSION_TYPE/lib/modules and will be loaded at boot time.

Additionally, the VETAR2A drivers for VMEbus platforms RIO4 and IPV have been developed especially for MBS usage. For CES/RIO4, the fast VTRANS memory mapping has been

\[2\] However, the kernel module wishbone.ko had to be slightly modified for kernel 4.9.30. This kernel version is not supported yet by the official white rabbit release.
implemented. For IPV, the fast extended local bus (ELB) mapping is used. These kernel mod-
ules are installed for the appropriate MBS systems at /mbs/driv/vetar* directories and will be
loaded at boot time.

1.6.3 User readout function

To read out the latched time stamp for each event, the user readout code f.user.c can call
standard functions of the etherbone library. Detailed examples are available on request. As a
course overview, the following calls are necessary in different parts of the readout code:

f_user_get_virt_ptr() All required handles are initialized here:

- The etherbone device handle /dev/wbm0 has to be opened with eb_socket_open() and
eb_device_open(). This handle is used in all subsequent calls.
- eb_sdb_find_by_identity has to be used to locate the GSI time latch unit (TLU) on
the etherbone bus. This handle is used for addressing the latched timestamp registers.

f_user_init() Here the TLU has to be prepared by several calls of eb_device_write() to appro-
priate wishbone registers:

- GSI_TM_LATCH.CH_SELECT - select the fifo channel
- GSI_TM_LATCH_FIFO_CLEAR - clear fifo contents
- GSI_TM_LATCH_TRIG_ARMSET - prepare the trigger timestamp latch

f_user_readout() The actual readout of the latched timestamp is done via etherbone "cycle" access:

- eb_cycle_open() to begin the io cycle of the etherbone bus
- eb_cycle_read() to check the fifo status and actually read the timestamp components
from the register GSI_TM_LATCH_FIFO_FTSHI (high word), GSI_TM_LATCH_FIFO_FTSLO
(middle low word), and GSI_TM_LATCH_FIFO_FTSSUB (sub low word).
- eb_cycle_write() to the register GSI_TM_LATCH_FIFO_POP to clear the fifo from the
old value.
- eb_cycle_close() to end the io cycle of the etherbone bus

The read out timestamp is further formatted and put into the subevent payload.
1.6.4 Direct TLU register access

Although the read-out of the latched timestamp by means of etherbone cycles works well, it has the disadvantage that the etherbone stack requires several bus accesses to retrieve a single data word from the timing receiver memory. This limits the maximum achievable eventrate performance.

To overcome this restriction, the standard firmware of the VETAR2A timing receiver has been extended\(^3\) with a special mode that allows to export any register space to a special address range on the VMEbus. So when the desired etherbone export address is set to the GSI TLU base (0x4000100), the TLU fifo registers are directly accessible in address mode V32 at the VME base address (0x50000000). This address is configured in the device driver as default for the VETAR slot number 5.

Switching between the standard etherbone mode (required for the tools eb-info etc.) and the direct access mode can be done on the fly by means of a control register that is made available as Linux sys filesystem entry by the vetar.ko kernel module (see 1.6.2). So the behaviour of the VETAR2A is simply adjustable from the user’s shell:

```bash
echo 0x4000100 >> /sys/class/vetar/vetar0/dactl
```

Writing this wishbone address for the export into the control register will switch on the direct access to the TLU.

```bash
echo 0xffffffff >> /sys/class/vetar/vetar0/dactl
```

Writing ”-1“ value to the register will switch back to etherbone mode.

This switching via sysfs handle is done directly from the MBS user readout code, depending on the mode that is defined at compile time. The provided examples will use the definition

\[\text{#define WR_USE_TLU_DIRECT 1}\]

To enable/disable either readout variant.

For the direct access mode, the user readout code has the following schema:

**f_user_get_virt_ptr()** The access mode is set and the VME registers are mapped:

- the value 0x4000100 is written to the file handle `/sys/class/vetar/vetar0/dactl` to select the TLU base for the exported registers
- the VME memory from base 0x50000000 is mapped to user space to access the TLU registers.
- auxiliary pointers are defined with appropriate register offsets from the mapped base address. This allows to directly read and write the TLU registers later

**f_user_init()** Preparation of the TLU is done by transparent io to the register pointer:

- \(\star ch\_select = WR\_TLU\_FIFO\_NR\) - select the fifo channel

\(^3\)Many thanks to Michel Reese from GSI CSCO!
• $\text{\texttt{\textchar"{}fifoclear = 0xFFFFFFFF\textchar"{}}} \text{ - clear fifo contents}$
• $\text{\texttt{\textchar"{}armset = 0xFFFFFFFF\textchar"{}}} \text{ - prepare the trigger timestamp latch}$

\texttt{f_user_readout()} The actual readout of the latched timestamp is also done via direct access of the mapped pointer.

Performance measurements with \texttt{MBS} on RIO4 and IPV VME systems prove the benefits of the direct access mode. The following table shows the accepted trigger rates for different set-ups:

<table>
<thead>
<tr>
<th></th>
<th>etherbone</th>
<th>direct tlu</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIO4:</td>
<td>21 kHZ</td>
<td>51 kHZ</td>
</tr>
<tr>
<td>IPV:</td>
<td>18 kHZ</td>
<td>42 kHZ</td>
</tr>
</tbody>
</table>

These values were measured without any other readout than the TLU.

The direct TLU access mode is currently available only for VETAR2A timing receivers. For other PCIe hardware such as PEXARIA, it is planned to implement a similair solution.

1.7 Mass storage interfaces

Since \texttt{MBS} version v6.2 the \textit{Remote File Input Output} (RFIO)\textsuperscript{4} API has been supported as mass storage interface. This has allowed two modes of operation:

• Inside GSI the \texttt{MBS} DAQ can write directly via socket into the tape robot (\texttt{GSTORE} tape archive) via dedicated data mover servers maintained by GSI IT.

• The \texttt{MBS} can write via sockets of a ”private“ RFIO server process on any Linux node to its regular file system. This allows for standalone \texttt{MBS} systems (”traveling DAQ“) to store the data to the disk of the Linux boot host machine.

In 2018 the GSI IT will no longer maintain the RFIO data movers. A new API \textit{Lightweight Tivoli Storage Manager} (LTSM)\textsuperscript{5} is going to replace the DAQ interface to the tape robot.

The \texttt{MBS} v6.3 will still support the RFIO interface for local disk servers, and also the GSI RFIO data movers in the migration phase. Additionally, a special local RFIO server is provided as a gateway to the LTSM archive.

In the following, the different use cases are explained. This refreshes the explanations given in the previous \texttt{MBS} release notes v6.2.

1.7.1 RFIO disk server

The actual RFIO server consists of two separate executables \texttt{rawDispRFIO} and \texttt{rawServRFIO} (see Figure ??). The first process \texttt{rawDispRFIO} (Disp for ”dispatcher“) is started by the user and

\textsuperscript{4}by Horst Göringer, GSI IT until 2016
\textsuperscript{5}by Thomas Stibor, GSI HPC, \url{https://github.com/tstibor/ltsm/}
may specify the port number for the RFIO socket to connect (default 1974). The second process rawServRFIO (Serv for "server") will be started anew for any MBS client on request. Both executables are provided on GSI Linux cluster at directory /cvmfs/it.gsi.de/gstore (for 64bit architectures with names rawDispRFIO64 and rawServRFIO64).

Important note: Since the installation path of these executables is not anymore constant at GSI, both programs must be copied to the same working directory with user write access!

The rawDispRFIO expects that the rawServRFIO is in the local directory and may write some auxiliary files there.

On the RFIO Linux node:

Before first usage, the executables must be copied to the working directory of the RFIO server. This can be in the home space of the example user "myaccount":

    cp /cvmfs/it.gsi.de/gstore/rawDispRFIO64 /u/myaccount/rfioserver
    cp /cvmfs/it.gsi.de/gstore/rawServRFIO64 /u/myaccount/rfioserver

(accordingly for the 32bit variants)

The server must be started in this directory:

Figure 1.4: Schematic overview of MBS data storage using the RFIO disk server.
myaccount@lxg0546:~$ cd /u/myaccount/rfioserver
myaccount@lxg0546:~rfioserver$ rawDispRFIO64 2345
-I- 64 bit RFIO server (lxg0546) listening on port 2345
    forks './rawServRFIO64' for each client request

On the mbs:

- first the connection to the RFIO server has to be established with MBS command `connect rfio < servername >:< port > -disk`
- any subsequent command `open file < filepathname >` will start writing to the RFIO server. The file path name must exist on the remote server host
- command `close file` will close the current file
- command `disconnect rfio` will shut down the disk server connection

This is illustrated in the following:

```bash
mbs> con rfio lxg0546:2345 -disk
-X86L-10:transport :Connected RFIO server!
```
```bash
mbs> open file /data.local1/myaccount/test_files_first=10 size=1000 -auto -rfio
-X86L-10:transport :open file at server lxg0546:2345::
-X86L-10:transport :/data.local1/myaccount/test_files_0010.lmd
-X86L-10:transport :-I- remote output file /data.local1/myaccount/test_files_0010.lmd closed
-X86L-10:transport :Rfio server lxg0546:2345: closed file after 999.981 MB, open next:
-X86L-10:transport :/data.local1/myaccount/test_files_0010.lmd
-X86L-10:transport :/data.local1/myaccount/test_files_0011.lmd
-X86L-10:transport :-I- remote output file /data.local1/myaccount/test_files_0011.lmd closed
-X86L-10:transport :Rfio server lxg0546:2345: closed file after 999.981 MB, open next:
-X86L-10:transport :/data.local1/myaccount/test_files_0011.lmd
-X86L-10:transport :/data.local1/myaccount/test_files_0012.lmd
mbs> close file
-X86L-10:transport :Rfio server lxg0546:2345: closed file after 332.956 MB written:
-X86L-10:transport :-I- remote output file /data.local1/myaccount/test_files_0012.lmd closed
mbs> disco rfio
-X86L-10:transport :-I- connection to data mover lxg0546 closed
-X86L-10:transport :Disconnected RFIO server!
```

The MBS RFIO commands don’t differ if either a 32 bit or a 64 bit Linux RFIO server is used.

### 1.7.2 Legacy RFIO tape server

Although the RFIO tape archive servers at GSI are going to be shut down in the near future, they are still supported by the MBS. The methods have not been changed since MBS v6.2.
Please note that the availability of a GSTORE archive account is mandatory (in the example below: archive \texttt{kurz}) and that all not existing directories, in the file path specified, will be created automatically by the RFIO server. The file path for the \texttt{open file} command must always be fully specified from its root (/\texttt{kurz}).

The example begins in a state where the \texttt{MBS} is already running and the acquisition has been started:

	exttt{mbs> connect rfio lxgstore -archive}
\texttt{-X86L-10:transport :I- successfully connected to data mover slxdm17:1994 (ATL server 1)}
\texttt{-X86L-10:transport :}
\texttt{-X86L-10:transport :Connected RFIO server!}
\texttt{mbs> open file /kurz/v62example/testfiles\_first=20 size=1000 -auto -rfio}
\texttt{-X86L-10:transport :open file at server lxgstore::}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0020.lmd}
\texttt{-X86L-10:transport :Rfio server lxgstore: closed file after 999.981 MB, open next:}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0020.lmd}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0021.lmd}
\texttt{-X86L-10:transport :Rfio server lxgstore: closed file after 999.981 MB, open next:}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0021.lmd}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0022.lmd}
\texttt{-X86L-10:transport :Rfio server lxgstore: closed file after 999.981 MB, open next:}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0022.lmd}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0023.lmd}
\texttt{mbs> close file}
\texttt{-X86L-10:transport :Rfio server lxgstore: closed file after 296.255 MB written:}
\texttt{-X86L-10:transport :/kurz/v62example/testfiles\_0023.lmd}
\texttt{mbs> disconnect rfio}
\texttt{-X86L-10:transport :I- connection to data mover slxdm17 closed}
\texttt{-X86L-10:transport :Disconnected RFIO server!}

1.7.3 RFIO gateway to new GSI storage interface LTSM

The new archiving interface LTSM requires the IBM Tivoli Storage Manager client libraries to be installed on the local host. However, these libraries are available for 64bit Linux systems only. Since \texttt{MBS} supports multiple different platforms, like LynxOS and Power PC Linux OS, it is not possible to deploy this storage interface on all of them. As a solution, a proxy gateway can be provided that is started on a system capable of TSM and may connected from any \texttt{MBS} node via the existing RFIO socket protocol (see Figure 1.5).

This gateway is implemented as a variation of the disk server (see section 1.7.1) and offers similar executables \texttt{rawDispLTSM64} and \texttt{rawServLTSM64}. At the moment, they are provided at the 64bit \texttt{MBS} Linux nodes at /\texttt{mbs/storage/PCx86/Linux\_3.2-64\_Deb/bin}. Later this service may be deployed to other nodes at GSI.
Figure 1.5: Schematic overview of MBS data storage using the RFIO gateway to LTSM.

**On the RFIO-LTSM Server node:**

As for the disk servers, the executables must be copied to the working directory of the RFIO-LTSM server. This can be in the home space of the example user "myaccount":

```bash
cp /mbs/storage/PCx86_Linux_3.2-64_Deb/bin/rawDispLTSM64 /u/myaccount/rfio2ltsm
cp /mbs/storage/PCx86_Linux_3.2-64_Deb/bin/rawServLTSM64 /u/myaccount/rfio2ltsm
```

The server must be started in this directory with the following usage available with option "-h":

```bash
X86L-59 myaccount> cd /u/myaccount/rfio2ltsm
X86L-59 myaccount> rawDispLTSM64 -h
```

```bash
rawDispLTSM64 for RFIO/MBS protocol
```
usage: rawDispLTSM64 ltsm_server ltsm_node password [portnumber]
X86L-59 myaccount > ./rawDispLTSM64 lxltsm01-tsm-server LTSM_TEST01 LTSM_TEST01 7799
Starting rawDispLTSM64 for LTSM server lxltsm01-tsm-server,
    LTSM node LTSM_TEST01, password:LTSM_TEST01, port:7799
    -I- 64 bit LTSM proxy server (X86L-59) listening on port 7799
         forks './rawServLTSM64' for each client request

Please note that LTSM server specifications and credentials must be set already on startup
of the RFIO gateway! These settings will be different for each experiment and may be requested
by the storage group of GSI IT department.

On the mbs:

- first the connection to the RFIO server has to be established with MBS command connect
  rfio < servername >:< port > -ltsm.

- any subsequent command open file < filepathname > will start writing to the RFIO server.
  The file path name must exist on the remote server host

- command close file will close the current file

- command disconnect rfio will shut down the disk server connection

This is very similar like for the disk server described in section 1.7.1:

mbs> connect rfio -ltsm x86l-59:7799
    -X86L-59:transport :Connected RFIO server!

1.8 New GSI boot servers with dedicated VLAN

A new server running Debian 8 (code name Jessie) is in operation since May 2017. The old
server has been shut down. The new server consists of two identical and independent hardware
servers. User harddisks are synchronized every 15 minutes between these two servers. System
harddisks and user harddisks of each of the servers are setup as RAID5 systems to further increase
the security and integrity of the services. In addition backups to the gsi tape library are done
every night.

The tasks of the new servers are the same as before and are mainly DHCP, TFTP and NFS
services for the diskless mbs nodes.

At the same date, when the new server went in to operation, all nodes served moved from GSI-
LAN into the Virtual Local Area Networks (VLAN) MBS-NETZ (mbs nodes for experiments)
or MBS-NETZ-ACC (nodes for accelerator tasks).
1.9 EPICS installations

The *Experimental Physics and Industrial Control System (EPICS)*[^1] control system framework version 3.14.12.5 has been installed at all **MBS** dedicated Linux systems of GSI, for X86, for IPV, and for RIO4 CPUs. For this reason the standard EPICS release distribution has been adjusted to the previously not supported RIO4 and IPV power PC platform.

The EPICS framework is available on all these platforms under the NFS mounted file system `/epics`. The environment can be set by sourcing the script `./epics/install/bin/epics` under bash shell.

Please note that EPICS is currently not supported for LynxOS platforms!

1.10 Miscellaneous

1. Improved Makefile structure: common `Makefile.config` to be included from user `Makefile`, support of multiple builds in parallel

2. New VTRANS memory mapping for RIO4 VME processor systems

3. **MBS** dedicated Linux systems provide installations of the analysis framework Go4 and the data acquisition framework DABC. They are available on the NFS mounted devices `/analysis` and `/dabc` after calling go4login or dabclogin, resp.

4. The GOSIP based pipelining ADC FEBEX4 is available now. Apart from the doubled sampling frequency of 100 MHz It features the same properties as the FEBEX3.

5. The slow control of **MBS** with the *Distributed Information Management system (DIM)* is not further supported anymore. Instead of this, it is recommended to use the new control and monitoring possibilities with the webserver via DABC, also for distributed event building systems.

6. Bugfix: Termination of **MBS** with key **Ctrl-Z** works now for all platforms. It is not necessary to kill explicitly all **MBS** processes by means of command `resl` when shutting down the DAQ. Additionally, the `rate` command keeps running after **Ctrl-Z** and resumes measuring the rates at next **MBS** startup.

List of Figures

1.1 Gosip control GUI for FEBEX front-end .................................................. 9
1.2 Schema of new MBS control channels ..................................................... 12
1.3 Webbrowser with MBS GUI ................................................................. 14
1.4 Processes and connections for RFIO disk server ....................................... 19
1.5 Processes and connections for RFIO gateway to LTSM ............................. 22
Bibliography


