GSI Data Acquisition System MBS
Release Notes V5.0

N. Kurz

October 11, 2006

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

1 MBS Release V5.0 .......................... 3
  1.1 General Remarks .......................... 3
  1.2 New Features in MBS Version 5.0 ................. 4
  1.3 Support of new VME Processor RIO4 ................ 4
  1.4 CAMAC MBS systems based on PC hardware .......... 5
    1.4.1 Hardware .......................... 5
    1.4.2 Software .......................... 6
    1.4.3 Performance ......................... 8
  1.5 RFIO data logging to disk server or GSI tape robot ....... 8
  1.6 Interactive VME server ....................... 10
  1.7 How to setup a multi-readout multi-event-builder (NxM) MBS system .... 12
    1.7.1 NxM setup specification ................ 13
    1.7.2 New MBS commands for NxM systems .......... 15
    1.7.3 Startup of NxM MBS systems ............... 15
  1.8 The AMUX ADC multiplexer system ................ 16
  1.9 The SIDEREM silicon microstrip readout system .......... 17
  1.10 Miscellaneous .......................... 17
    1.10.1 MASTER and CONTROLLER_ID summary ....... 17
    1.10.2 MBS print_m function .................. 18
    1.10.3 Time sorter (m_to) improvement ............ 19
    1.10.4 LynxOS version 4.0 and MBS version 5.0 ...... 19
    1.10.5 VC32-CC32 Esone Server .................. 19
Chapter 1

MBS Release V5.0

1.1 General Remarks

This Release Notes can be retrieved as pdf file from the web page: http://daq.gsi.de/document.htm

They cover all changes for MBS version 4.3 to 5.0. Please use this release notes in conjunction with the MBS user manual and the release notes V4.3, also available on the MBS homepage.

Since October 17th, 2006, the MBS version 5.0 has become the new production version. MBS version 5.0 replaces the old production version 4.3 as well as the intermediate versions 4.4 and 4.5. 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. In case of troubles don’t hesitate to contact N.Kurz, GSI Tel.: 2979, E-Mail: N.Kurz@gsi.de

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

2. Make sure that this command is not followed by another command like mbslogin v43 (or v30, v41, v42, v44, v45), source /mbs/develogin.com, or source /mbs/oldlogin.com.

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 LynxOS prompt.

4. All user functions and programs (m_read_meb, m_collector) have to be recompiled and linked! Your current make files are still valid and make clean followed by make in your MBS working directory is sufficient to rebuild your programs.
5. We strongly recommend to rename your main scripts to startup and and shutdown your MBS to startup.scom and shutdown.scom. Otherwise the MBS graphical user interface doesn’t work.

1.2 New Features in MBS Version 5.0

The new MBS version 5.0 provides several new features which will be described in further sections.

1. Support of new VME Processor RIO4
2. CAMAC MBS systems based on PC hardware
3. RFIO data logging to disk server or GSI tape robot
4. Interactive VME server
5. How to setup a multi-readout multi-event-builder (NxM) MBS system
6. The AMUX ADC multiplexer system
7. The SIDEREM silicon microstrip readout system
8. Miscellaneous

MBS version 5.0 supports the following processor platforms and LynxOS versions:

LynxOS 2.5: CVC, E7, RIO2, PC
LynxOS 3.1: RIO3
LynxOS 4.0: RIO4, PC

1.3 Support of new VME Processor RIO4

The next generation RIO4 of the VME PowerPC processors from the company CES has been integrated into the MBS. It is equipped with an on board copper gigabit ethernet interface and runs on LynxOS version 4.0.

Care has been taken, that from the user side everything shall be set or done as on the RIO3. Especially the static mapping for VME access has been made identical as on the rio3. Please see the table below and also release notes V4.3. Also the function calls for dynamic VME address mapping (find_controller) and vme block data transfers (bma_xy calls) follow the identical syntax.

Only the MASTER keyword in the MBS setup.usf file has to be changed and set to 12, with respect to setup files for RIO3 systems.

Performance improvements of 25 to 50% are measured compared to the RIO3. I.e. single cycle VME a32 d32 accesses show for RIO3 4.6 MB/s, for RIO4 5.6 MB/s, reading from a standard VME memory.
Unfortunately, at editing time of this note, the RIO4 is only available for VME64X crates with 5 rows P1/J1, P2/J2 and P0. CES promised to have a version for standard VME crates ready by the end of 2006.

**RIO2 VME static mapping:**

<table>
<thead>
<tr>
<th>Address Modifier</th>
<th>VME Hardware Address</th>
<th>Map Address</th>
<th>Map Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM: 0x09 (A32)</td>
<td>0x0</td>
<td>0xE00000000</td>
<td>0xE000000</td>
</tr>
<tr>
<td>AM: 0x39 (A24)</td>
<td>0x0</td>
<td>0xEE000000</td>
<td>0x10000000</td>
</tr>
<tr>
<td>AM: 0x29 (A16)</td>
<td>0x0</td>
<td>0xEFFE0000</td>
<td>0x10000</td>
</tr>
</tbody>
</table>

**RIO3 and RIO4 VME static mapping:**

<table>
<thead>
<tr>
<th>Address Modifier</th>
<th>VME Hardware Address</th>
<th>Map Address</th>
<th>Map Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM: 0x09 (A32)</td>
<td>0x0</td>
<td>0x50000000</td>
<td>0x10000000</td>
</tr>
<tr>
<td>AM: 0x39 (A24)</td>
<td>0x0</td>
<td>0x4F000000</td>
<td>0x10000000</td>
</tr>
<tr>
<td>AM: 0x29 (A16)</td>
<td>0x0</td>
<td>0x4E000000</td>
<td>0x10000</td>
</tr>
<tr>
<td>AM: 0x0B (BLT, D32 Block)</td>
<td>0x0</td>
<td>0x50000000</td>
<td>0x10000000</td>
</tr>
<tr>
<td>AM: 0x08 (MBLT, D64 Block)</td>
<td>0x0</td>
<td>0x60000000</td>
<td>0x10000000</td>
</tr>
</tbody>
</table>

### 1.4 CAMAC MBS systems based on PC hardware

The old CVC CAMAC controller is running more than a decade within the MBS data acquisition system and is still supported. Based on a 68030 Motorola processor running LynxOS it appears now to be slow compared to modern processors available today. Furthermore, the CVC can neither be repaired nor rebuild anymore. Therefore a successor system based on “standard” PC hardware, running also the real-time operation system LynxOS has been developed.

#### 1.4.1 Hardware

This new CAMAC MBS system is composed of a PC equipped with the GTBP (GTB to PCI) interface. Up to 15 GTBC (GTB to CAMAC, see MBS release notes V4.3) controller may be connected in a chain to the GTBP via GTB cables. The last GTB has to be properly terminated. The GTB is a bus protocol, which connects electronic devices over distances of more than hundred meters for controls and data transfers. Each DAQ system must contain a CAMAC crate with crate id 1. All other CAMAC crates must have a unique crate id ranging from 2 to 15. The CAMAC crate id can be set by a rotary switch on the GTBC controller. The CAMAC crate with id 1 must contain a GSI CAMAC trigger module in slot number 1. Trigger inputs and deadtime output shall be handled in the same way as with the CVC based MBS systems.
trigger input signal will cause a CAMAC LAM on the CAMAC bus and is routed via the GTBC - GTB cable - GTBP into the interrupt system of the PC.

Summarizing it can be stated, that the new system shall be setup similar as a CVC system. The GTBC replaces both, CVC and CBV as CAMAC controller, the GTB cables replace the VSB cables and the PC together with the GTBP replaces the processing part of the CVC.

The GTB bus, the GTBP and the GTBC controller were developed by Jan Hoffmann from the EE department of GSI.

**1.4.2 Software**

**Interrupt driver and - handler:**
Based on LynxOS, a driver has been developed for the handling of interrupts from the CAMAC trigger module. This driver and the appropriate device are loaded at boot time.

**ESONE server:**
If a proper MBS setup.usf file is supplied the well known CAMAC ESONE server can be started and used immediately in the MBS framework. It works as with the CVC or the supported VME processors. Of course only CAMAC crate identifiers from 1 to 15 can be addressed.

**f_user.c:**
No table readout is supported, but template user readout functions (f_user.c) are provided, which make it very easy to create a valid f_user.c from an existing readout table. If an old readout function used with CVC or VME processor shall be reused, no or very small changes are necessary. Ask N.Kurz for details.

**setup.usf:**
Some parameters of the MBS setup file need to be modified, but templates are also available. See below an example of the setup parameters, which need to be modified and some comments for them. In this example four remote CAMAC crates are defined and readout. All parameter arrays shown in this example represent parameters as a function of the crate identifier, ranging from 0 to 15.

```
MASTER = 9, -

REM_CAM_BASE   = (0x0, 0xdc380000, 0xdc580000, 0xdc780000, 0xdc980000, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0), -
REM_CAM_LEN    = (0x0, 0x50000, 0x50000, 0x50000, 0x50000, 0x0, 0x0, 0x0, 0x0, 0x0), -
```
The MBS recognizes a PC as master if: \textbf{MASTER} = 9.

There is only a remote ESONE window, since there is no local CAMAC crate (as with the CVC). The value of the \textbf{REM_ESONE_BASE} parameter is depending at which base address the PCI system automatically assigns the so called PCI “local space 0” of the GTBP PCI card. This address assignment depends on the PCI configuration of a PC. A PCI scan has to be made once after mounting the GTBP (ask N.Kurz how to do it). The GTBP maps the PCI “local space 0” transparently to the GTB bus. The GTBC itself was designed to function identical as the old CBV CAMAC controller. It decodes the lower 24 address bits send by the GTBP and translates in the usual way addresses into the crate id, station number, subaddress and function. CAMAC offset is still 0x180000, crate offset is still incremented by 0x200000 for each crate id. In the above example the PCI “local space 0” was found to be 0xdc000000. For the \textbf{REM_ESONE_BASE} the CAMAC offset of 0x180000 has to be added, which gives 0xdc180000.

The remote CAMAC crate base addresses \textbf{REM_CAM_BASE} are derived then easily by: \textbf{REM_ESONE_BASE} + (0x2000000 * crate id).

Setting \textbf{REM_CAM_LEN} to 0x50000 maps the complete CAMAC functional space.

The base address of the MBS sub-event pipe is specified with \textbf{LOC_PIPE_BASE}. It depends on the RAM size of the PC, as the pipe shall be inside the RAM.

The readout \textbf{RD_FLAG} set to 1 tells the system to readout this crate for each accepted trigger.
The CONTROLLER_ID for crate 0 has to be set again to 9 (PC) and for crate 1 to 4 the CONTROLLER_ID of the GTBC is 5. CBV and GTBC are from the users point of view identical and have the same CONTROLLER_ID.

In one of the CAMAC crates a GSI trigger module has to be plugged in station number one for handling the trigger requests to the MBS system. This is assigned with the TRIG_STAT_NR parameter.

startup procedure:
The only difference compared to other single processor MBS systems in the startup procedure is the parameter crate=1, indicating that the trigger module is in that crate. See example below.

```
start task m_util
load setup setup.usf crate=1
c1 da co
clear trig_mod
set trig_mod
start task m_read_meb "./m_read_meb"
start task m Collector
start task m_transport
start task m_esone_serve
```

1.4.3 Performance
The basic CAMAC access of the PC-GTBP-GTBC system is not faster than CAMAC accesses to the local crate with a CVC. One CAMAC read operation needs about 2 \(\mu\)s on both systems. But whereas the event processing overhead for small events on a CVC is 500 \(\mu\)s, it is only 50 \(\mu\)s seconds on the new PC system. This results in a gain of a factor 10, 20000 events/sec, compared to 2000 events/sec with a CVC for small events. Also for larger events the new PC system is substantially faster, since event-building, data sending over the network and taping is of course much faster.

1.5 RFIO data logging to disk server or GSI tape robot
RFIO (Remote File Input/Output) is a TCP based protocol and a library to write data directly to a remote mass storage device. The functions provided have the same functionality as the standard C disk IO calls. At GSI an MBS system can write list mode data directly to a data server and from this MBS version 5.0 it is also possible to write data directly into the GSI tape
robot.

Please note, that the syntax for RFIO data logging to disk servers changed (on Linux and MBS) compared to MBS version 4.3.

**RFIO data logging to disk servers:**

To start the RFIO server on a Linux node type:

```plaintext
> rawDispRFIO
```

First open a RFIO connection with the `MBS` command:

```plaintext
mbs> connect rfio nodename -disk
```

To file MBS data on this server use the `MBS` command:

```plaintext
mbs> open file /fullpath/fn -rfio
```

Or another example:

```plaintext
mbs> open file /fullpath/fn size=500 first=1 -auto -rfio
```

Close files as usual:

```plaintext
mbs> close file
```

At the very end of data logging, close the RFIO connection to the disk server with:

```plaintext
mbs> diconnect rfio
```

**RFIO data logging to GSI tape robot:**

First open a RFIO connection to a (free) data mover of the GSI tape robot with the `MBS` command:

```plaintext
mbs> connect rfio gsitsma -arch
```

To file MBS data on the tape robot use the `MBS` command:

```plaintext
mbs> open file /fullpath/fn -rfio
```

Or another example:

```plaintext
mbs> open file /fullpath/fn size=500 first=1 -auto -rfio
```

Close files as usual:
mbs> close file

At the very end of data logging close RFIO connection to the tape robot with:

mbs> disconnect rfio

Note:
Before starting data logging to the tape robot, a valid tape archive must exist. If a directory, specified in the path name is not existing, it will be created by the RFIO system. All characters in the path name must be in lower case for data writing to the tape robot.

Note:
At GSI at maximum 43 characters are allowed including full path -, file name and the “.lmd” extension, for both data logging to disk servers or to tape robot. If the “.lmd” extension is omitted in the “file open” command, it will be appended automatically by the MBS system. With the -auto flag of the MBS “file open” command, the number of available characters is reduced by 4.

Note:
If errors occur during RFIO related MBS commands, or data logging, the MBS system might disconnect automatically from the RFIO server. Please connect again to your RFIO server in these cases, according to the syntax described above.

Performance measurements with one MBS event builder PC running LynxOS have shown a sustained data logging speed of 10-11 MB/sec, reaching the bandwidth of the fast ethernet interfaces in use. This is about a factor of two better, than what we get with the DLT 8000 drives.

Note:
The on-line data flow through the GSI network might create congestion for other network devices (depending, where sender and receiver are located).

Note:
When writing data to a “private” Linux server the data is not save against disk crashes. It must be moved per user command (admscli) to the final place in the robot.

The RFIO functions have been provided by H.Goeringer from the IT department.

1.6 Interactive VME server

A new server program for interactive read and write operations on VME in single cycle mode is available now. It can be started either standalone under LynxOS by calling “vme”, or in an MBS session with the command “start task m_vme_serv”. It can be started on E7, RIO2, RIO3 and RIO4 and hides all differences (and difficulties) of these platforms from the user. To stop the
vme server in stand alone mode type quit or exit.
It allows accesses (read, write) in A16 (VME short), A24 (VME standard) and A32 (VME extended) address modes. Apart from A16, all these address modes allow VME accesses with data sizes D16 and D32 possible.
Commands to this VME server can be grouped as usual in .scom script files, to program more complicated sequences. This allows i.e. easy startup of VME controlled devices, like high voltage power supplies.

The commands to read from VME are:
vr16d16, vra24d16, vra24d32, vra32d16, vra32d32

The commands to write to VME are:
vwa16d16, vwa24d16, vwa24d32, vwa32d16, vwa32d32

The syntax is identical for all read commands (example: vra32d16):

VRA32D16 addr [size repeat] -LOG -NOPRINT -FC

PURPOSE (m_vme_serv) VME read A32 D16
PARAMETERS
addr VME address of slave module to read from
size size of data to read (in bytes)
repeat repetition count of VME read
-LOG log result in mbs log file
-NOPRINT don’t print result on terminal
-FC if RIO2 or RIO3 use find_controller function instead of static VME mapping

The syntax is also identical for all write commands (example: vwa32d32):

VWA32D32 addr data [size repeat] -LOG -NOPRINT -FC -INC

PURPOSE (m_vme_serv) VME write A32 D32
PARAMETERS
addr VME address of slave module to write to
data data to write
size size of data to write (in bytes)
repeat repetition count of VME write
-LOG log result in mbs log file
-NOPRINT don’t print result on terminal
-FC if RIO2 or RIO3 use find_controller function instead of static VME mapping
-INC increment data by 1 per write cycle
Note: All commands, parameters and flags can be in lower or upper cases.
Note: The use of the repeat parameter together with the -NOPRINT flag allows for easier work with oscilloscopes.

1.7 How to setup a multi-readout multi-event-builder (NxM) MBS system

Multi-event-builder MBS systems (in short NxM systems) are supported since MBS version 4.0, but have not been described in detail yet. These kind of MBS systems are used in cases, where a single event-builder is either not capable to receive or send the amount of data to data storage or online analysis purposes. A working example of such a system is the MBS system of the FOPI experiment. FOPI uses 8 VME readout processors and 5 event-builder PCs.

In such a system the readout processors are connected via the GSI trigger bus as usual. To be able to build complete events, each event-builder connects at startup phase to each readout processor. Although some address mapped data connections between readout processor and event-builder, like Block VSB, Block VME and PVIC are implemented it turned out, that the message oriented TCP network protocol is the most versatile and therefore the only one used. TCP has also the big advantage, that it can be used with 10Mbit -, 100Mbit - or Gigabit ethernet networks without any software change necessary. Another advantage is the fact, that the data connections are automatically created, since all LynxOS processors are mandatory connected to the ethernet and work as so called “diskless” nodes. Performance can only be gained in such a system if all event-builders can receive data simultaneously. This is enabled by the fast switched network technology at GSI. As all switches also have network bridge functionality, it can be easily arranged, that no other network node disturbs or block the data traffic between readout processors and event-builders.

It may also be noted, that this MBS topology supports also the “delayed event building” mode, where data is send possibly in macroscopic beam pauses, which leaves the full power of a readout processor for acquiring data from its connected digitizers. See release notes version 4.3 for details.

In single - or multi processor MBS systems, connected with address mapped buses, the m_collector process was responsible for reading data from the sub-event pipes and for the event-building. Since TCP is a message oriented protocol, two new MBS processes (m_ds, m_dr) replace m_collector. The data sender process m_ds runs on the readout processor and the data receiver process m_dr runs on the event-builder. m_ds reads sub-events from the local sub-event pipe and tries to dispatch and send the data in a round robin scheme to the event-builders, which have themselves declared ready for data receiving to the m_ds. The m_dr processes reads data from all readout processors and builds events for further data logging and online monitoring.

In the process of simultaneous data sending to the event-builders the protocol implemented guarantees that no data (sub-event) is lost, and that the sub-events are not mixed into different
events.

TCP via network is not very efficient for small data sizes. To circumvent this problem, the user specifies in the setup file, described later in this section, how many fragments shall be send in one protocol cycle. Since a readout processor can read more than one crate and produce therefore more than a single sub-event, the entity is called fragment. To summarize: A fragment stems from one readout processor and is composed of one or more sub-events. To minimize the TCP response time, the TCP_NODELAY option is enabled on all TCP sockets in use.

1.7.1 NxM setup specification

The topology of such a system is described in the set_mo.usf setup file. In order to illustrate its content an example of a system with 3 readout processors and 2 event-builders will be shown in the following:

set_mo.usf:

```
MO_SETUP: -
  DS_HOSTNAME_0   = "R2F-5", -
  DS_HOSTNAME_1   = "R3-12", -
  DS_HOSTNAME_2   = "R3-40", -
#    |    # DS node index
#  DR_HOSTNAME_0   = "X86-8", -
  DR_HOSTNAME_1   = "X86G-3", -
#    |    # DR node index
#  # DS node index: 0 1 2
#    |    |    |    |
  RD_PIPE_TRANS_MODE_0 = (0x1001, 0x1001, 0x1001, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), -
  RD_PIPE_TRANS_MODE_1 = (0x1001, 0x1001, 0x1001, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), -
#    |    # DR node index
#  # FORM_MODE    = 1, -
  N_FRAG         = 19, -
  MAX_EVT_SIZE   = 40000, -
#    |    # DR node index
#  # EV_BUF_LEN    = (0x8000,0x8000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), -
```
N_EV_BUF = ( 50, 30, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), -
N_STREAM = ( 10, 10, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0), -
OUT_MODE = ( 1, 1, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)

With **DS_HOST_NAME**_0** to **DS_HOST_NAME**_15** a list of readout processors is specified. The keywords **DS_HOST_NAME**_x** must be listed consecutively from **DS_HOST_NAME**_0** to the maximum number of readout processors. In the example above with 3 readout processors: 0, 1, 2.

With **DR_HOST_NAME**_0** to **DR_HOST_NAME**_15** a list of event-builders need to be specified. The keywords **DR_HOST_NAME**_x** must be listed consecutively from **DR_HOST_NAME**_0** to the maximum number of event-builders. In the example above with 2 event-builders from 0 to 1.

The transfer mode has to be specified with **RD_PIPE_TRANS_MODE**_x**, where **x** denotes the event-builder index given in the event-builder list **DR_HOSTNAME**_x**. Each event-builder has to read from all readout processors, therefore for each event-builder a list of data transfer modes for all readout processors must be specified. In the example above all transfer modes are set to 0x1001, which sets TCP sockets for data transfer. As mentioned above some other transfer modes are implemented and could be set freely in the ds - dr matrix, but practically only TCP is used by the experiments.

**FORM_MODE** = 1 is the only supported mode yet.

**N_FRAG** specifies the number of fragments (sub-events), which shall be send in one protocol cycle. This circumvents TCP performance problems with small data sizes. The **N_FRAG** parameter shall be optimized by changing its value for short periods under real conditions until a maximum throughput is found. Above a fragment size of 1 KByte, the **N_FRAG** value is not so critical and it is in most cases sufficient to check the values 1, 5, 10, 100, 500. Events with trigger types 12, 13 (used for delayed event-building) and trigger type 15 (stop acquisition command) always cause the immediate transport of the fragments, regardless of the value of **N_FRAG**.

The maximum event size in bytes has to be specified with **MAX_EVT_SIZE**. A decent safety margin is recommended.

For the formatted event output, the output buffer length **EV_BUF_LEN**, the number of buffers in a stream **N_EV_BUF** and the number of streams **N_STREAM** have to be specified for each event-builder. These parameters follow the same rules as in the conventional setup.usf files. See MPrS user manual. It is recommended to set **EV_BUF_LEN** to its maximum possible value of 0x8000 bytes. It is observed that with **N_EV_BUF** set to 50 and **N_STREAM** set to 10, the performance is in an optimal bandwidth.

**OUT_MODE** has to be set to 1 for each event-builder.
1.7.2 New MBS commands for NxM systems

disable receive_data
An event-builder can be disabled on the fly during data taking. A reason could i.e. be a broken output device.

   enable receive_data
An event-builder can be on the fly enabled to receive data.

   set dr_flushtime time
Flushes output data, after “time” seconds, although output stream is not completely full.

1.7.3 Startup of NxM MBS systems

In the above described 3x2 NxM MBS system, startup - and shutdown procedures (.scom scripts) for all processors are needed. In the following examples of startup scripts for readout processors and event-builders are shown. It is assumed that the 3 readout processor environments are in the directory vme_0, vme_1 and vme_2:

start_readout_0.scom:

sta m_util
load setup ./vme_0/setup.usf
load mo_setup ./set_mo.usf
set trig_mod -multi
enable irq
sta m_read_meb ./vme_0/m_read_meb
sta m_ds

start_readout_1.scom:

sta m_util
load setup ./vme_1/setup.usf
load mo_setup ./set_mo.usf
set trig_mod -slave
enable irq
sta m_read_meb ./vme_1/m_read_meb
sta m_ds

start_readout_2.scom:

sta m_util
load setup ./vme_2/setup.usf
load mo_setup ./set_mo.usf
set trig_mod -slave
enable irq
sta ta m_read_meb ./vme_2/m_read_meb
sta ta m_ds

start_event-builder.scom:

sta ta m_util
load mo_setup ./set_mo.usf
sta ta m_dr
sta ta m_transport
sta ta m_stream_serv
set str 1
sta ta m_daq_rate
set dr_flush 1

Please note that a common startup script is sufficient for the event-builders. All these individual startup scripts can be called in a common startup.scom, to make the mbs startup simple.

startup.scom:

set dispatcher r2f-5
@./start_readout_0
set dispatcher r3-12
@./start_readout_1
set dispatcher r3-40
@./start_readout_2
#
set dispatcher x86-8
@./start_event-builder
set dispatcher x86g-3
@./start_event-builder
#
set dispatcher r2f-5
start acq

Similar scripts to shut down the MBS system can be created easily.

1.8 The AMUX ADC multiplexer system

This section is only intended to draw attention, and not meant as a detailed description.

The AMUX system is in operation since about two years at the SHIP experiment and has, remarkably, not shown a failure up to now.

The AMUX module, designed by J. Hoffmann (hardware) and W. Ott (DSP software), is a NIM module and serves for several purposes. Mainly it reads up to 4 of the SILENA high resolution
spectroscopy NIM ADCs and has additional functionalities as scaler, pattern unit and various time measurement capabilities. For control and data transfer it is equipped with a GTB port.

The AMUX MBS system is build up with a RIO2/3 processor, a GSI VME trigger module and a GSI SAM3 module in a VME crate. The SAM3 is connected with its GTB port to the AMUX boards used. At SHIP 11 AMUXs are in a single GTB control and readout chain. In addition, a local trigger bus connects all AMUXes with the SAM3.

Event triggers are generated internally from any ADC detecting first a valid signal. This trigger is then routed via its local trigger bus to the SAM3, which itself triggers the GSI VME trigger module for readout. Up to 400 events can be stored already in each AMUX at the very front end, which minimizes deadtime and makes the system very efficient. Care has been taken to optimize performance and guarantee the synchronization of sub-events.

In the meantime this system has been successfully cloned for the new TASCA experiment at the UNILAC.

If interested, please ask J.Hoffmann, W.Ott or N.Kurz for details.

1.9 The SIDEREM silicon microstrip readout system

This section is only intended to draw attention, and not meant as a detailed description of this new SIDEREM system.

This new system powers, controls and reads the silicon microstrip detector, developed by the AMS collaboration. It works as a standalone MBS - or as a MBS sub-system in a larger MBS data acquisition. It has been successfully used with 4 microstrip detectors the first time in the S271 experiment in August 2006 at the FRS at GSI.

Each double sided microstrip detector consists of 640 strips on the front - and 384 crossed strips on the back side. The SIDEREM modules, housed in NIM crates, the heart of this new system was developed by J.Hoffmann (hardware) and W.Ott (DSP software) of the EE department of GSI. One SIDEREM module connects to one microstrip detector and supplies all power lines, as well as all signal lines needed for control and data transfer. It is equipped with a GTB interface, which allows for up to 16 SIDEREM modules in a single readout chain.

The SIDEREM MBS (sub)system is build up with a RIO3 processor, a GSI VME trigger module and a GSI SAM3 module in a VME crate. The SAM3 connects with its GTB port to the SIDEREMs used. In addition, a local trigger bus connects all SIDEREMs with the SAM3. Care has been taken to optimize performance and guarantee sub-event synchronization.

If interested, please ask J.Hoffmann, W.Ott or N.Kurz for details.

1.10 Miscellaneous

1.10.1 MASTER and CONTROLLER_ID summary

With respect to previous versions, some new processors and controllers have been introduced to the MBS. If used, they have to be specified in the MBS setup.usf file with the MASTER and
CONTROLLER_ID keywords. A complete list is given below:

**MASTER:**
- CVC: 1
- E6: 2
- E7: 3
- RIO2: 8
- PC: 9
- RIO3: 10
- RIO4: 12

**CONTROLLER_ID:**
- CVC: 1
- E6: 2
- E7: 3
- AEB-EBI: 4 (not supported anymore)
- CBV, GTBC: 5
- CVI: 6
- CAV: 7
- RIO2: 8
- PC: 9
- RIO3: 10
- VC32-CC32: 11
- RIO4: 12

**Note:**
The supported FASTBUS controller NGF is not listed above. The reason is, that a small VME crate is integrated in this controller. Depending on which processor is plugged into this VME crate, the MASTER has to be set accordingly.

### 1.10.2 MBS `printm` function

It often occurs, that software developments, or parts of it, are intended to be used inside and outside of the MBS context. In multi processor MBS systems the problem arises, that calls to the standard printf C function are not seen on the MBS control terminal. This arises due to the fact, that such MBS systems are controlled by only one processor connected to a terminal. On the other hand the standard MBS message system, which overcomes this problem, is only available in the MBS context and not in standalone applications. Also, the syntax for MBS message calls is somehow cumbersome.

From this version a function `printm` is available. It follows exactly the same syntax as the well-known printf function. Compiled with the standard MBS compile flag `GSI_MBS (-DGSI_MBS)` it routes all printouts (printf) automatically to the MBS message system. Omitting this compile flag, simply the printf function is called. In addition, it shall be very easy to change from printf to printm and vice versa with standard editors. It also gives the possibility to change old code easily from printf to printm.
To access printm, a function prototype f_ut_printm.h residing in /mbs/prod/inc has to be included and the code linked against the lib_mbs.a library.

Use \texttt{LIB = $(MBSROOT)/lib$(GSI\_CPU\_PLATFORM)/lib_mbs.a} in Makefiles. Of course the C code \texttt{f\_ut\_printm.c} can also be copied from the /mbs/prod/src directory to local ones.

1.10.3 Time sorter (m_to) improvement

A small, but alleviative change was made in the time sorter program (m_to). See release notes 4.3 for details of the m_to functionality. Before this version event input connections, to be time sorted, could not be disconnected when no event was coming in. The data sender TCP socket simply got no notice of a broken socket and waited for the next event (and with the next event it would have detected the disconnected socket).

Now disconnects can be made at any time regardless of the state of the data sender.

1.10.4 LynxOS version 4.0 and MBS version 5.0

From MBS version 5.0, PCs and RIO4 are supported running the newest LynxOS version 4.0.

The most concerning change from older LynxOS versions with respect to 4.0 is the removal of the widely used memory mapping functions \texttt{smem\_create}, \texttt{smem\_get} and \texttt{smem\_remove}. They are replaced by standard POSIX calls \texttt{open}, \texttt{mmap}, \texttt{shm\_open} and \texttt{shm\_unlink}, with quite different syntaxes. To make life easier wrapper functions for the smem_ functions have been developed. This allows to call the old smem_ functions with their wellknown syntax. Inside the MBS context this is done automatically. Outside of the MBS context it can be used by linking against the lib_mbs.a library.

1.10.5 VC32-CC32 Esone Server

In MBS version 4.3 a short discription was made how to use the VC32-CC32 VME to CAMAC controller from WIENER. Now the MBS ESONE server also works with this controller. See detailed manual on MBS webpage.
Contents