**RIO4: system memory - and VME mapping**

**PPC_Mon: boot_size:**

The parameter boot_size in the NVRAM of the RIO4 specifies the maximum memory size seen by the operating system (LynxOS, Linux). It can be set in the monitor program PPC_Mon. If set to 0 or 0x20000000 the whole memory (512 MB) is used by the system. With boot_size some additional steering can be done:

-----------------------------------------------

If boot_size is set up to (at maximum) 0x10000000 (256 MB) two direct mappings for memory and VME are activated at kernel startup:

1) Direct address page 0xe0000000 points to memory address 0x0. This page has a size of 256 MB (0x10000000 bytes) and covers the first half of the memory.

2) Direct address page 0xf0000000 points to VME address 0x0 for A32 accesses (address modifier: 0x9). This page has also size of 256 MB and covers therefore the VME address range from 0x0 to 0x0ffffffc

For both cases the direct mapping provides the fastest access modes possible on this hardware. I.e. the single cycle VME A32 speed is ~ 70 % faster than with other mapping procedures. See below.

The covered space can be directly accessed by adding the memory or VME address to the appropriate page base. The disadvantage of using the direct VME page: No bus error occurs if one accesses a VME address, with no related hardware. Instead the processor simply gets stuck.

-----------------------------------------------

If boot_size is set to values larger than 0x10000000, only direct memory mapping is activated at kernel startup:

1) Direct address page 0xe0000000 points to memory the address 0x0. This page has a size of 512 MB (0x20000000 bytes) and covers the whole memory.

The usage of this direct memory mapping is the same as described above.

Attention: In this setting all programs relying on direct (and fast) VME A32 mapping will fail without notification!

-----------------------------------------------

Especially check LOC_PIPE_TYPE, LOC_PIPE_BASE and LOC_PIPE_SEG_LEN when working with the MBS. The boot_size parameter can be checked with "/bin/ces/hwconfig show boot_size" in a running LynxOS system.

-----------------------------------------------
Memory mapping:

1) Physically non-consecutive memory mapping:

a) malloc:
With the malloc function only ~ 33MB in total can be allocated on RIO3 (LynxOS 3.1) and RIO4 (LynxOS 4.0) in a process. Anyhow, malloc can not be used as an inter process method.

b) Process shared memory: Mapping with the shm_open, ftruncate and the mmap function:
In LynxOS 3.1 and earlier versions, shared memory for inter process communications was accomplished with the smem_get function. This function is not available any more for LynxOS 4.0 and therefore for the RIO4.
The same functionality can be achieved, by using the functions shm_open, f_truncate and mmap.
To allow an easy migration for legacy applications a wrapper function has been developed and put in the MBS sources from version 5.0 onwards. This function has the identical name smem_get and follows the identical syntax as the original smem_get function.
To utilize it, the application either has to be linked against the mbs_lib.a library or the file f_smem.c has to be included in the compile chain.
This mapping procedure can also be utilized with PCs running LynxOS 4.0.

2) Physically consecutive memory mapping:

a) Direct mapping as explained above on address offset 0xe0000000

b) Process shared consecutive memory: Mapping with the /dev/mem device and mmap
In LynxOS 3.1 and earlier versions, consecutively mapped shared memory for inter process communications was accomplished with the smem_create function. This function is not available any more for LynxOS 4.0 and therefore for the RIO4.
The same functionality can be achieved, by opening the /dev/mem device and the mmap function.
To allow an easy migration for legacy applications a wrapper function has been developed and put in the MBS sources from version 5.0 onwards. This function has the identical name smem_create and follows the identical syntax as the original smem_create function.
To utilize it, the application either has to be linked against the mbs_lib.a library or the file f_smem.c has to be included in the compile chain.
This mapping process can also be utilized with PCs running LynxOS 4.0.
VME address space mapping:

On the RIO3 512 pages of 4 MB each, add up to 2 GB of VME address space, which can be mapped simultaneously. On the RIO4 this size has been reduced to 256 pages of 4 MB, which provides a total space of 1 GB.

On the RIO4, this (1) GB has to be distributed between static – and dynamic VME mapping. See below:

1) Static VME mapping:

a) Direct mapping as explained above for A32 (AM: 9).

b) To allow an easy way to map VME address space, static pages have been configured on the RIO3 and on RIO4 in an identical scheme for single cycle accesses.

Disadvantage: Statically mapped VME space cannot be freed and is fixed to its (limited) VME address range and access type (address modifier).

The static maps have been generously sized for GSI applications, but they appear to be too big in order to leave enough space for flexible dynamic VME mapping on the RIO4. As an acceptable compromise the static mapping for A32 (AM = 9) accesses has been reduced from 256 MB (0x10000000) to 96 MB (0x06000000). The static map for MBLT (D64) VME block transfer has been removed completely. With this solution most legacy programs developed on the RIO3 work also on the RIO4. See VME static pages for RIO3 and RIO4 below:

<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>0x09 (A32)</td>
<td>0x0</td>
<td>0x500000000</td>
<td>0x100000000</td>
</tr>
<tr>
<td>0x39 (A24)</td>
<td>0x0</td>
<td>0x4f0000000</td>
<td>0x10000000</td>
</tr>
<tr>
<td>0x29 (A16)</td>
<td>0x0</td>
<td>0x4e0000000</td>
<td>0x10000</td>
</tr>
<tr>
<td>0x0B (BLT)</td>
<td>0x0</td>
<td>0x500000000</td>
<td>0x10000000</td>
</tr>
<tr>
<td>0x08 (MBLT)</td>
<td>0x0</td>
<td>0x600000000</td>
<td>0x10000000</td>
</tr>
</tbody>
</table>

RIO3 static mapping:

RIO4 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>0x09 (A32)</td>
<td>0x0</td>
<td>0x500000000</td>
<td>0x100000000</td>
</tr>
<tr>
<td>0x39 (A24)</td>
<td>0x0</td>
<td>0x4f0000000</td>
<td>0x10000000</td>
</tr>
<tr>
<td>0x29 (A16)</td>
<td>0x0</td>
<td>0x4e0000000</td>
<td>0x10000</td>
</tr>
</tbody>
</table>

Up to LynxOS 3.1 the mapping for single cycle VME access could be done with the smem_create function and the map address and segment size as input parameters. On RIO4 and LynxOS 4.0, either the dev/mem device and mmap to map the segment or the smem_create wrapper can be used as described in section "Process shared consecutive memory". VME Block transfer aspects see below.
2) Dynamic VME single cycle transfer mapping:

Up to RIO3, LynxOS 3.1, dynamic mapping was done with the find_controller function. It turns out, that find_controller works only if the VME address range requested is inside the static mapping, which makes it useless for dynamic VME mapping.
A set of new functions for VME single cycle mapping are provided by CES for the RIO4 (bus_open, bus_map, bus_unmap and bus_close).
Since its usage is not too user friendly, two functions (f_map_vme_rio4, f_unmap_vme_rio4) contained in the file f_map_vme_rio4.c have been developed and can be found under LynxOS in the directory /nfs/groups/daq/usr/kurz/rio4/busmap. With these functions, a template main program to show its syntax and a Makefile VME single cycle mapping on the RIO4 shall be simple. It is planned to move f_map_vme_rio4.c and its prototype into the MBS sources after some extensive testing.
In addition a running MBS environment, which makes usage of this mapping method, can be found on /nfs/groups/daq/usr/kurz/mbstest/rio4/rio4_busmap_sicy_vme.
Within this MBS setup it has furthermore been tested, that the following PPC_Mon and MBS setup.usf parameters work fine. See also direct memory mapping explained above:

<table>
<thead>
<tr>
<th>PPC_Mon</th>
<th>setup.usf</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot_size</td>
<td>LOCPIPE_TYPE</td>
</tr>
<tr>
<td>0x4000000</td>
<td>1</td>
</tr>
<tr>
<td>0x8000000</td>
<td>1</td>
</tr>
<tr>
<td>0x10100000</td>
<td>1</td>
</tr>
</tbody>
</table>

3) Dynamic VME Block transfer mapping:

As mentioned in 1), the static maps for VME block transfer have been removed on the RIO4, in order to free space for dynamic VME mapping.
For dynamic VME block mapping a new function xvme_map is provided by CES. A full functioning MBS environment using dynamic mapping for single cycle VME and the xvme_map function for dynamic VME block mapping can be found under LynxOS in: /nfs/groups/daq/usr/kurz/mbstest/rio4/rio4_busmap_bma_rd.
This MBS setup can be used as template for BLT, BLT and the faster 2eVME and 2eSST VME block readout, if the VME slave modules provide these access modes.
Various VME slaves have been tested: SAM3/5 (BLT, MBLT), VME memory from Micro Memory Inc (BLT), CAEN V878 (BLT, MBLT) and a RIO4 as VME slave (BLT, MBLT, 2eSST).
Especially 2eSST transfers from RIO4 to RIO4 achieve speeds of up to 150 MB/s and are the fastest VME mode, supported by MBS VME hard- and software.
All three pipe settings shown above in 2) have been tested and work with VME block transfers.
(Very) large MBS sub-event pipes, (very) large consecutive shared mapped memory:

If an application needs very big shared memory for either fast online storage of data, or the MBS setup needs a large sub-event pipe, the following shall be done and considered:

The memory for LynxOS shall be set to a minimum of 0x4000000 (64 MB) with the boot_size parameter. With this setting 0x1c000000 (448 MB) are at maximum available for user (shared and consecutive) mapping. Since boot_size is set below 0x10000000, fast direct single cycle A32 VME is enabled within the restriction described above, but also static and dynamic (single cycle and block) VME mapping can be used. Fast direct memory mapping can only be used for pipes smaller or equal 0x10000000 (256 MB). Therefore this mapping procedure cannot be used in this case. Instead the memory needs to be mapped with the /dev/mem device and the mmap function, or the wrapper function smem_create. See 2) in "Memory mapping" above. Please note that memory mapped with this procedure can be accessed with a lower speed than with direct memory mapping, but this shall come only into effect for fast VME block transfers (MBLT, 2eSST) into this memory.

In /nfs/groups/daq/usr/kurz/mbstest/rio4/rio4_dimap_sicy_mmap_pipe a running MBS setup, which uses a 448 MB sub-event pipe and direct or static VME single cycle mapping can be found.

In /nfs/groups/daq/usr/kurz/mbstest/rio4/rio4_bus_map_bma_rd_mmap_pipe a running MBS setup, which uses dynamic VME single cycle and - VME block readout together with a 448 MB sub-event pipe can be found.

Thank you for your enduring attention!