DLD: PMC pm8001 Forensic Debug

Primary Contact: <mark_salyzyn@us.xyratex.com>

References

- HLD.pm8001-forensic
- 2090556_spc_fw_forensic_appnote_063051.pdf or later
- 20800222_pm8001_pm_081181.pdf or later
- Pulled-in Reference Material highlighted in yellow. Please refer to the up to date references for absolute details. Any distribution of this document with this content must also comply with PMC Non-Disclosure requirement. Other non-highlighted content may be viewed as derivative.

Abstract

See HLD.pm8001-forensic

Describe the technical itemized details of the driver interaction with debugfs tree in sysfs; along with namespace and logic specifications.

Definitions

None

Forensic handler mechanics

The driver will disable all forensic features if the kernel define CONFIG_SCSI_PM8001_DEBUG_FS is set to 0. All pm8001 drivers as shipped by Xyratex will have this feature enabled by default, but it is expected that a community push of the forensic feature will by default be disabled in the Kconfig. The mechanism for handling CONFIG_SCSI_PM8001_DEBUG_FS as undefined is part of the mechanism to differentiate community builds from Xyratex added-value builds. In the future there may be a need for an internal define override so add support for _CONFIG_SCSI_PM8001_DEBUG_FS to act as such.

If discovered during coding and testing, the forensic tree will need to contain individual enable-ment if there is a memory or processing overhead associated with the individual feature. We do not forsee a need for enable-ment except for the Event Logging Level and Event Logging Size at this time.

Script examples assume that the results are placed in /tmp; the implementer of the forensic diagnostic dump feature may wish to place the results in the /var/log/ tree as a possible permanent location for debug snapshots to comply with LSB. Such mechanisms are up to the Trinity Team's management tools and for the purposes of this document outside the scope since it crosses to another group.

Base controller debugfs node for the forensics is in the controller forensic node hierarchy at /sys/kernel/debug/pm8001/pm8001.X/forensic/ where X is an unique hba number identifier. All references below refer to the above directory as .../forensic/ as shorthand.

Please provide the GSM memory region dumps in following GSM Memory Regions Dump in either a binary or an ASCII file format. If providing a memory dump in ASCII-coded hexadecimal format, please structure the file output so that 8 DWORDs appear per row with 64K per block.

Host GSM Memory Regions Dump

Since there is little or no memory or processing overhead associated with supporting the Host GSM Memory Regions Dump, this forensic tree is always present and available. Base controller debugfs node for the GSM memory region dumps is in the controller

forensic node hierarchy in a sub-directory tree .../forensic/1.gsm_memory/. These register dumps are referenced off BAR3. The large amount of data is to be provided in Binary format (pending feasibility) and is expected to take on the following tree nodes:

- 0.io_status 256KB @ offset 0x640000
- 1.rb_storage 128KB @ offset 0x68000
- 2.rb_pointers 4K @ offset 0x8A1000
- 3.rb_access 4K @ offset 0x8A8000
- 4.gsm_sm 1M @ offset 0x400000

It is expected the data can be retrieved with:

```
# mkdir /tmp/1.gsm_memory &&
for i in forensic/1.gsm_memory/* ; do
        cat ${i} > /tmp/1.gsm_memory/${i##*/} ;
done
```

Since binary content can have no text descriptive headers, it is the responsibility of the script to place the content by name as an identifier.

Alternate Implementation

If binary retrieval is either not feasible, or is difficult, then an utf8 representation may be required. It is expected to retrieve the values using the following user-space scripting instead:

```
# cat forensic/1.gsm_memory/*/* > /tmp/1.gsm_memory
```

The above memory dumps are in binary and are read in 1KB utf8 converted chunks from each of the following nodes as labelled below:

- 0.io_status/0x00000000
- 0.io_status/0x00000400
- . . .
- 0.io_status/0x0003FC00
- 1.rb_storage/0x00000000
- 1.rb_storage/0x00000400
- . . .
- 1.rb_storage/0x0001FC00
- 2.rb_pointers/0x00000000
- 2.rb_pointers/0x00000400
- 2.rb_pointers/0x00000800
- 2.rb_pointers/0x00000C00
- 3.rb_access/0x00000000
- 3.rb_access/0x00000400
- 3.rb_access/0x00000800

- 3.rb_access/0x00000C00
- 4.gsm_sm/0x00000000
- 4.gsm_sm/0x00000400
- . . .
- 4.gsm_sm/0x000FFC00

If this approach is taken, the descriptive headers need to be part of the first entry. When coding for these fields consider the simplest approach (binary or text).

Host GSM Memory Regions Dump Operation

In detail, we provide five dump files of the GSM memory regions as listed in the table below: IO Status Table (IOST), Ring Buffer Storage (RB_storage), Ring Buffer Pointers (RB_pointers), Ring Buffer Access (RB_access), and GSM 1M Shared Memory (GSM_SM).

The shift address in column 4 is the address that is loaded into the MEMBASE-III inbound
window shift register to allow host process access of the specified memory region. The use of this register is explained in detail see [1], Section 2.6.1 of the PM8001 Tachyon SPC 8x6G Programmers Manual (PMC-2080222).

Table - GSM Memory Regions

Memory Region	Size	Offset	Shift Address
IO Status Table	64KB	0x0000 to 0xFFFF	0x64_0000
IO Status Table	64KB	0x0000 to 0xFFFF	0x65_0000
IO Status Table	64KB	0x0000 to 0xFFFF	0x66_0000
IO Status Table	64KB	0x0000 to 0xFFFF	0x67_0000
RB Storage 0~64K	64KB	0x0000 to 0xFFFF	0x68_0000
RB Storage 64K~128K	64KB	0x0000 to 0xFFFF	0x69_0000
RB Pointers	4KB	0x1000 to 0x1FFF	0x6A_0000
RB Access	4KB	0x8000 to 0x8FFF	0x6A_0000

GSM SM0	64KB	0x0000 to 0xFFFF	0x40_0000
GSM SM1	64KB	0x0000 to 0xFFFF	0x41_0000
GSM SM2	64KB	0x0000 to 0xFFFF	0x42_0000
. . .			
GSM SM16	64KB	0x0000 to 0xFFFF	0x4F_0000

Input/Output Queue

In order to support this class of forensic data, the base controller debugfs node for the IQ and OQ are placed in the directory tree .../forensic/2.queue/. Within that directory one will find the node tree:

- iq/iomb/00/0.header (optional holder of the sub-header if simplifies coding)
- iq/iomb/00/000 (individual sub-header, followed by data)
- iq/iomb/00/001 (data)
- . . .
- iq/iomb/00/XXX
- iq/iomb/01/0.header (optional holder of the sub-header if simplifies coding)
- iq/iomb/01/000 (individual sub-header, followed by data)
- iq/iomb/01/001
- . . .
- iq/iomb/01/XXX
- . . .
- iq/iomb/XX/XXX
- oq/iomb/00/0.header
- oq/iomb/00/000
- . . .
- oq/iomb/XX/XXX
- unlock (optional atomic completion, for the future)

adjusted for the specific limits of the driver (number of ports, queue depth).

To acquire the data blob:

```
# find 2.queue -type f | xargs cat > /tmp/2.queue
```

It is noted that the forensic document uses hex for ci and pi, but decimal for the IOMB index. I suggest we use decimal for the reference in debugfs, and hex indexes inside the content for the build-in header. This is consistent since [<hex>] is used elsewhere as a

means of denoting an address.

Note: find reports in filesystem order. The driver must make sure the above alphanumeric order (POSIX Bourne Shell expansion) is preserved. If this is not possible, then an appropriate ‘sort’ action needs to be placed into the pipeline. Since the content for debugfs is typically filled-in at open, and not at read operations, the driver could take actions to try to fill them all in atomically; towards that goal it is expected that a read of iq/header (or any read) could lock the content, and oq/iomb/<lastq#>/<lastq> or unlock, to unlock it, with an implied timeout. This is a future-proofing suggestion and today is not a requirement. The script should abide by this requirement despite the current acceptance that these values remain dynamic.

This should also be observed for casual use. For example, if one reads iq/iomb/01/230, one should read unlock if it is there if they are done.

Alternate Implementation

Look into a binary representation if the dump can be made more atomic as it is retrieving dynamic content. Since unstructured binary data is not human readable, and the boundaries of each queue entry would not be clear, binary representation is not the preference. When coding for these fields consider the simpler approach. Besides the dump would not look like the expectations in the forensic documentation (below) further removing binary as a preference.

Input/Output Queue Dump Details

Please provide dumps of all the inbound and outbound queues that are in use on your SPC 8x6G controller in an UTF8 hexadecimal format shown below. The SPC 8x6G controller uses Inbound Queues (IQs) to receive messages from the host. Outbound Queues (OQs) are used to transfer messages to a host. IQs and OQs are circular queues located in host memory. Depending on the host use model, the host can program up to 32 IQs and 32 OQs. See [1] Section 2.3, “Inbound Queues” and Section 2.4, “Outbound Queues” for complete queue descriptions.

Please dump all elements of each queue to the depth defined in the queue’s configuration table. Please do not truncate the queue dump to an arbitrary size that is less than the queue size specified in the queue configuration table.

The example shows a queue created to service Port 0 only. There should be a queue dump file for each queue in use where [X] = the queue number.

IQ[X] Details

depth = 0x100

```
latest ci = 0xf6
latest pi = 0x61
IOMB[0]:
0x81002006 0xd21c28b0 0x00000001 0x00000208 0x00000102 0x00000000 0x00000000 0x00020000
0x9dd20908 0x00000001 0x00000000 0x00000000 0xad89a000 0x00000000 0x00000208 0x00000000
IOMB[1]:
0x81002006 0xa26d2870 0x00000002 0x00000208 0x00000102 0x00000000 0x00000000 0x00000000
0x57d30908 0x00000001 0x00000000 0x00000000 0xad89a000 0x00000000 0x00000208 0x00000000
:
:
:
IOMB[254]:
0x81002006 0x327826b0 0x00000009 0x00000208 0x00000102 0x00000000 0x00000000 0x00000000
0x19d80908 0x00000001 0x00000000 0x00000000 0xad89a000 0x00000000 0x00000208 0x00000000
IOMB[255]:
0x81002006 0xe26c26b0 0x00000009 0x00000208 0x00000102 0x00000000 0x00000000 0x00000000
0x1ad80908 0x00000001 0x00000000 0x00000000 0xad89a000 0x00000000 0x00000208 0x00000000
```

Note: For consistency, the array index will be provided as a hex value (eg: IOMB[FF]) in the content. This will match with latest ci and latest pi from the sub-header. Since each entry is individually captured and is not atomic, the user must be prepared for this. One way of checking for an atomic pull is to reaquire iq/header to see if the ci or pi have changed.

GSM Registers

Read 4 bytes at a time, in the specified order, with a hierachy root at .../forensic/3.gsm/. At most 1K can be dumped per node as converted to HEX. For the quick size estimation below, format is space separated, 6-hex address, followed with up to 4 8-hex data representations (53 formatted bytes for every 16 bytes or 4DW of data). An estimate is made of the number of content bytes returned from each node. This represents the audit to ensure that the node limit of 4096 characters is not typically exceeded (a reasonable limit on output). A descriptive header to describe the content source should be provided and is not part of the estimated length.

Script to retrieve the complete dump:

```
# cat forensic/3.gsm/* > /tmp/3.gsm
```

An audit of the size of each element to determine feasibility, content and format follows.

The final format selected must be consistent with other forensic dumps, using either 0xXXXXX or [XXXXX] (5 hex digits) for the address field should be OK (from a parser standpoint) and can be followed with up to 8 0xXXXXXXXX values, but given the fragmentation it may be better to limit to 4 as used in the size estimate. Comfortable human readability is a priority. Many of the examples of dump output in the forensic documents place the address in [XXXX] format so that is the likely expectation.

Please provide register dumps of the GSM registers in the format shown below.

Notes:

1. The registers are sparse. Make sure that host does not blindly read a range of addresses. If the address is not defined, the SPC MIPS CPU will hit a fatal bus exception error.
2. Register Dump: 4-byte reading each time.
3. Strictly following the order below.
4. All values listed are in hexadecimal.
5. Values enclosed in brackets [0000] refer to a memory address.
6. Individual addresses that aren't part of an address range are shown as [0000]:
7. Consecutive memory address ranges are shown as [0000] - [00B8].
8. All Offsets below are referenced off of BAR3

SPC

.../01.SPC
[0x000000-0x0000FF] 64DW
[0x050000-0x05002F] 12DW
1484 bytes to represent

BDMA

.../02.BDMA
[0x010000-0x01000B] 3DW
[0x010010-0x010013] 1DW
[0x010020-0x010073] 25DW
[0x010084-0x01008B] 2DW
[0x010090-0x010097] 2DW
[0x01009C-0x0100A3] 2DW
[0x0100A8-0x0100AF] 2DW
[0x0100D0-0x0100D7] 2DW
[0x0100F0-0x0100FB] 3DW
[0x010100-0x01010B] 3DW
[0x010110-0x01011B] 3DW
[0x010120-0x01012B] 3DW
[0x010130-0x01013B] 3DW
[0x010140-0x01014B] 3DW
[0x010150-0x01015B] 3DW
[0x010160-0x01016B] 3DW
[0x010200-0x01027F] 32DW
[0x010300-0x010303] 1DW
[0x010340-0x010343] 1DW
1355 bytes to represent

PCIe APP

.../03.APP

[0x013000-0x01300B] 3DW
[0x013010-0x01304F] 16DW
[0x013060-0x01306B] 3DW
[0x013070-0x0130BB] 19DW
[0x013100-0x01317B] 31DW
[0x013200-0x01327B] 31DW
[0x013280-0x0132A3] 9DW
[0x013300-0x013323] 9DW
[0x013328-0x01339F] 30DW
[0x0133C0-0x0133C7] 2DW
[0x013400-0x0134FF] 64DW
[0x013800-0x013803] 1DW
2929 bytes to represent

PCIe PHY

../04.PHY
[0x014000-0x01401B] 7DW
[0x014020-0x014027] 2DW
[0x01402C-0x01402F] 1DW
[0x014040-0x014057] 2DW
[0x01405C-0x01405F] 1DW
[0x014064-0x01406F] 3DW
[0x0140A0-0x0140B3] 5DW
[0x0140C0-0x0140DF] 8DW
[0x014100-0x014127] 10DW
[0x014130-0x01413F] 4DW
[0x014180-0x014183] 1DW
[0x0141B0-0x0141BF] 4DW
[0x0141D0-0x0141E7] 6DW
[0x0141EC-0x0141EF] 1DW
[0x0142F8-0x014327] 12DW
[0x014330-0x01433F] 4DW
[0x014380-0x014383] 1DW
[0x0143B0-0x0143BF] 4DW
[0x0143D0-0x0143E7] 6DW
[0x0143EC-0x0143EF] 1DW
[0x0143F8-0x014427] 12DW
[0x014430-0x01443F] 4DW
[0x014480-0x014483] 1DW
[0x0144B0-0x0144BF] 4DW
[0x0144D0-0x0144E7] 6DW
[0x0144EC-0x0144EF] 1DW
[0x0144F8-0x014527] 12DW
[0x014530-0x01453F] 4DW

[0x014580-0x014583] 1DW
[0x0145B0-0x0145BF] 4DW
[0x0145D0-0x0145E7] 6DW
[0x0145EC-0x0145EF] 1DW
[0x0145F8-0x014627] 12DW
[0x014630-0x01463F] 4DW
[0x014680-0x014683] 1DW
[0x0146B0-0x0146BF] 4DW
[0x0146D0-0x0146E7] 6DW
[0x0146EC-0x0146EF] 1DW
[0x0146F8-0x014727] 12DW
[0x014730-0x01473F] 4DW
[0x014780-0x014783] 1DW
[0x0147B0-0x0147BF] 4DW
[0x0147D0-0x0147E7] 6DW
[0x0147EC-0x0147EF] 1DW
[0x0147F8-0x014827] 12DW
[0x014830-0x01483F] 4DW
[0x014880-0x014883] 1DW
[0x0148B0-0x0148BF] 4DW
[0x0148D0-0x0148E7] 6DW
[0x0148EC-0x0148EF] 1DW
[0x0148F8-0x01494B] 21DW
3422 bytes to represent

PCIe CORE

.../05.CORE
[0x018000-0x018047] 19DW
[0x018050-0x01805F] 4DW
[0x018070-0x01808B] 7DW
[0x018094-0x0180B7] 9DW
[0x018100-0x01812B] 11DW
[0x019010-0x019027] 6DW
[0x019030-0x019033] 1DW
780 bytes to represent

OSSP

.../06.OSSP
[0x020000-0x02001B] 7DW
[0x020028-0x02002F] 2DW
[0x02003C-0x020043] 2DW
[0x020050-0x020057] 2DW
[0x020064-0x02006B] 2DW

[0x020078-0x02007F] 2DW
[0x02008C-0x020093] 2DW
[0x0200B0-0x0200DF] 12DW
[0x0200F0-0x0200FF] 4DW
[0x020120-0x020127] 2DW
[0x020220-0x020227] 2DW
[0x020320-0x020327] 2DW
[0x020420-0x020427] 2DW
[0x020520-0x020527] 2DW
[0x020620-0x020627] 2DW
[0x020720-0x020727] 2DW
[0x020820-0x020827] 2DW
[0x020908-0x02090B] 1DW
[0x020910-0x020917] 2DW
[0x020920-0x020927] 2DW
[0x020930-0x02093F] 4DW
1037 bytes to represent

SSPA

.../07.SSPA
[0x032000-0x032033] 13DW
[0x032074-0x0320AB] 14DW
[0x0320B8-0x0320CF] 6DW
[0x0320E0-0x0320E3] 1DW
[0x036000-0x036033] 13DW
[0x036074-0x0360AB] 14DW
[0x0360B8-0x0360CF] 6DW
[0x0360E0-0x0360E3] 1DW
[0x03A000-0x03A033] 13DW
[0x03A074-0x03A0AB] 14DW
[0x03A0B8-0x03A0CF] 6DW
[0x03A0E0-0x03A0E3] 1DW
[0x03E000-0x03E033] 13DW
[0x03E074-0x03E0AB] 14DW
[0x03E0B8-0x03E0CF] 6DW
[0x03E0E0-0x03E0E3] 1DW
[0x042000-0x042033] 13DW
[0x042074-0x0420AB] 14DW
[0x0420B8-0x0420CF] 6DW
[0x0420E0-0x0420E3] 1DW
[0x046000-0x046033] 13DW
[0x046074-0x0460AB] 14DW
[0x0460B8-0x0460CF] 6DW
[0x0460E0-0x0460E3] 1DW

[0x04A000-0x04A033] 13DW
[0x04A074-0x04A0AB] 14DW
[0x04A0B8-0x04A0CF] 6DW
[0x04A0E0-0x04A0E3] 1DW
[0x04E000-0x04E033] 13DW
[0x04E074-0x04E0AB] 14DW
[0x04E0B8-0x04E0CF] 6DW
[0x04E0E0-0x04E0E3] 1DW
4048 bytes to represent

HSST

.../08.HSST
[0x021000-0x02104B] 19DW
254 bytes to represent

Note: a decision was dropped to join HSST, HSST(A) and HSST(B) into one because the size would exceed 4K. A decision that was applied to all to ensure that the sequence of data is in the same order as the forensic document.

LMS_DSS

.../09.LMS_DSS
[0x030000-0x03006B] 27DW
[0x034000-0x03406B] 27DW
[0x038000-0x03806B] 27DW
[0x03C000-0x03C06B] 27DW
1440 bytes to represent

SSPL_6G

../10.SSPL_6G
[0x031000-0x03105B] 23DW
[0x031060-0x03107F] 8DW
[0x0310A0-0x0310AB] 3DW
[0x035000-0x03505B] 23DW
[0x035060-0x03507F] 8DW
[0x0350A0-0x0350AB] 3DW
[0x039000-0x03905B] 23DW
[0x039060-0x03907F] 8DW
[0x0390A0-0x0390AB] 3DW
[0x03D000-0x03D05B] 23DW
[0x03D060-0x03D07F] 8DW
[0x031DA0-0x03D0AB] 3DW
1820 bytes to represent

HSST(A)

.../11.HSST
[0x033000-0x03301B] 7DW
[0x033020-0x033067] 22DW
[0x0330B4-0x0330E7] 13DW
[0x037000-0x03701B] 7DW
[0x037020-0x037067] 22DW
[0x0370B4-0x0370E7] 13DW
[0x03B000-0x03B01B] 7DW
[0x03B020-0x03B067] 22DW
[0x03B0B4-0x03B0E7] 13DW
[0x03F000-0x03F01B] 7DW
[0x03F020-0x03F067] 22DW
[0x03F0B4-0x03F0E7] 13DW
2368 bytes to represent

LMS_DSS(A)

.../12.LMS_DSS
[0x040000-0x04006B] 27DW
[0x044000-0x04406B] 27DW
[0x048000-0x04806B] 27DW
[0x04C000-0x04C06B] 27DW
1440 bytes to represent

SSPL_6G(A)

../13.SSPL_6G
[0x041000-0x04105B] 23DW
[0x041060-0x04107F] 8DW
[0x0410A0-0x0410AB] 3DW
[0x045000-0x04505B] 23DW
[0x045060-0x04507F] 8DW
[0x0450A0-0x0450AB] 3DW
[0x049000-0x04905B] 23DW
[0x049060-0x04907F] 8DW
[0x0490A0-0x0490AB] 3DW
[0x04D000-0x04D05B] 23DW
[0x04D060-0x04D07F] 8DW
[0x041DA0-0x04D0AB] 3DW
1820 bytes to represent

HSST(B)

.../14.HSST
[0x043000-0x04301B] 7DW
[0x043020-0x043067] 22DW
[0x0430B4-0x0430E7] 13DW
[0x047000-0x04701B] 7DW
[0x047020-0x047067] 22DW
[0x0470B4-0x0470E7] 13DW
[0x04B000-0x04B01B] 7DW
[0x04B020-0x04B067] 22DW
[0x04B0B4-0x04B0E7] 13DW
[0x04F000-0x04F01B] 7DW
[0x04F020-0x04F067] 22DW
[0x04F0B4-0x04F0E7] 13DW
2368 bytes to represent

MBIC IOP

.../15.MBIC_IOP
[0x060000-0x0600BB] 47DW
[0x0600C0-0x0600C3] 1DW
[0x0600C8-0x0600CB] 1DW
[0x0600D0-0x0600F3] 1DW
[0x0600D8-0x0600DB] 1DW
[0x0600E0-0x0600E3] 1DW
[0x0600E8-0x0600EB] 1DW
[0x0600F0-0x0600F3] 1DW
[0x0600F8-0x0600FB] 1DW
[0x060100-0x06012B] 11DW
[0x060130-0x06013B] 3DW
[0x060140-0x060167] 10DW
[0x060180-0x06019F] 8DW
[0x060400-0x06040B] 3DW
[0x060410-0x060507] 62DW
2094 bytes to represent

MBIC AAP1

.../16.MBIC_AAP1
[0x070000-0x0700BB] 47DW
[0x0700C0-0x0700C3] 1DW
[0x0700C8-0x0700CB] 1DW
[0x0700D0-0x0700F3] 1DW
[0x0700D8-0x0700DB] 1DW
[0x0700E0-0x0700E3] 1DW
[0x0700E8-0x0700EB] 1DW

[0x0700F0-0x0700F3] 1DW
[0x0700F8-0x0700FB] 1DW
[0x070100-0x07012B] 11DW
[0x070130-0x07013B] 3DW
[0x070140-0x070167] 10DW
[0x070180-0x07019F] 8DW
[0x070400-0x07040B] 3DW
[0x070410-0x070507] 62DW
2094 bytes to represent

SPBC

.../17.SPBC
[0x090000-0x09002F] 12DW
[0x09003C-0x09004B] 4DW
[0x09005C-0x09006F] 5DW
[0x09007C-0x09008F] 5DW
[0x09009C-0x0900AF] 5DW
[0x090100-0x0901A3] 41DW
[0x0901C0-0x09024B] 35DW
[0x090260-0x0902EB] 35DW
[0x090360-0x09036B] 3DW
[0x091014-0x091033] 8DW
[0x091054-0x091073] 8DW
[0x091094-0x0910A7] 5DW
[0x091400-0x09141F] 8DW
[0x091800-0x09181F] 8DW
2538 bytes to represent

GSM

.../18.GSM
Description (as opposed to a lengthy and useless list):

- Not contiguous, only ‘even’ offsets 0 and 8, not 4 or C.
- Data address range [0x070000-0x0723FB]
- holes at 0x070010-0x07001C, 0x070030-0x07006C, 0x0700A0, 0x0700B0-0x0700FC, 0x070500-0x0707FC, 0x070C40-0x070FFC, 0x071400-0x0717FC, 0x071C00-0x071FFC
- Every pair of 0x07???0,0x07???8 counts as a single utf8 line of 42 characters.

Lines=576-1-4-5-48-60-64-64=330
13860 bytes to represent total, must divide into 4 parts in order to cover scope if placed on sysfs. This forces us to use debugfs to deal with the content if not split. This content size of a few K is not insurmountable requiring any splitting of switching to a binary format.

Message Unit Register Capture

Placed at .../forensic/4.msgu.

Script to retrieve the complete dump:

```
# cat forensic/4.msgu > /tmp/4.msgu
```

The 64-bit BAR specified at PCI configuration address 0x10 (MEMBASE-I) contains the following doorbell and scratchpad registers:

Offset:
[0004-0007]
[0020-0023]
[0040-0077]

Event Log

Event Logging already has some mechanics built-in to report the event logs as hba nodes iop_log and aap_log. These interfaces are broken in that they only report the header (documented in 'Event Log Operation' below) and the entry[0] in raw hex utf8 format and no other. In addition, it breaks from the sysfs one-element-one-line-one-entry standard. Hardly useful for forensic or human consumption. The log is also hard-coded in size making it not suitable for forensics which may demand a larger size on duplication to catch a detailed reference to the trouble report.

Legacy Event Log

There is a need to enhance the existing mechanism for regular use, allowing a variable-sized log and to be able to, in a human-readable utf8 and sysfs-style action incrementally pull the event log independent of the forensic output. Since the existing mechanism has had little use and is not entirely useful, we will change the format so that the new mechanism is obvious, more useful to the uninitiated and somewhat orthogonal in format to the original.

There is an insmod parameter logging_level that needs to be respected, we will need to add another called logging_size. These will be used to set the default.

A read-only node of aap1_log and iop_log exists to retrieve individual eldest-first log entries reporting in a space-separated list of values:

- Event Severity Level hex value (1-5)
- Timestamp as a 1ns resolution epoch timer in decimal (0-14757395258967641292 7 nano seconds)

- Unique event sequence number in decimal (0-4294967295)
- Hex sequence of words referencing the event details (00000000-FFFFFFFF …) as limited by the length.
- Provision for additional text payload should the driver in the future provide log interpretation (not to be added as part of this rewrite).
- Trailing newline

This format is chosen to make it more human-readable. Nothing specifies the utf8 format as expected by PMC.

When there are no more events in the queue, either the latest/last valid entry is returned requiring the application that is reading it to compare or an empty (0 byte) line is returned. By returning the latest entry in perpetuity we embody the expectations from this logging node, but have added depth to it by reporting history.

Any invalid entries (eg: size 4 or less) will be skipped to find a valid entry. A partial read of the debugfs node will truncate and thus discard remaining data, the expected minimum buffer size for a read should be 56 bytes in order to guarantee no loss of crucial data. No provision has been made to support multiple workers pulling the values out of the event log. Once the event log is full, the older entries are dropped, it is the responsibility of the caller to pull the entries off in a timely manner. Changing the log size results in undefined behavior regarding the content of the logs (leaving the software author the flexibility), and may or may not clear the log contents. The only defined way to clear the log is to repeatably read the log until no more entries are available.

To retrieve the event log as a snapshot using a script:

```
# for i in /sys/class/scsi_host/host*/*_log ; do
        l=""
        while read j < ${i}; do
                [ -n "${j}" -a "${j}" != "${l}" ] || break ;
                echo "${j}" ;
        done >/tmp/5.event.${i#*/} ;
done
```

The result will be a /tmp/5.event.aap_log and /tmp/5.event.iop_log files filled with the events dump in utf8.

The mechanism to support the incremental read of the event log will require the driver to maintain an internal consumer index to match up to the producer and consumer(begin) indexes (and their associated range to check validity) as managed by the hardware.

Another consideration is that the driver has it's own existing event log mechanism with the sole intent of reading entry 0. We will need to resize as necessary for the purposes of the full event log dump.

PMC may differ in their expectations of the log dump format. A script to possibly translate the events into a PMC-expected (hex) readable format:

```
# awk '{
        printf ("%08x %08x %08x %08x %s %s %s %s\n", \
                $1, $2/8/4294968296, $2/8%4294968296, \
                $3, $4, $5, $6, $7);
}' /tmp/5.eventlog.aap1
```

demonstrates that we can alter the format to comply with their needs, in user space. Determining size of the entry is possible, but beyond the scope of this simple demonstration.

Regardless, this format is not suitable for a simple one-piece retrieval that is desirable for a forensic collection, we will provide a separate forensic binary format for simplified retrieval. The above only serves to document an alternate means of retrieving the log.

debugfs Event Log

Base controller debugfs node for the event log is in the controller node hierarchy and is located in a directory tree .../forensic/5.eventlog/. Within that directory one will find two read-write nodes: 0.level and 1.size. One will also find two read-only nodes 3.aap1 and 4.iop that are the internal binary representations of the two associated event logs. Binary is chosen because the user may select a large size

0.level can interactively set the event log to one of the documented levels from 0 to 5, with a default of 1 (minimal). 1.size an be interactively set the event log sizes in number of bytes, aligned on a 32 byte boundary, from 0 (disabled) to a system-imposed limit (8M might be an appropriate limit to impose initially), with a default of 131072 (128K or 4095 entries plus one header). There is no effort to differentiate the size and log level for the two available event logs.

To retrieve the event log as a snapshot using a script:

```
# for i in forensic/5.eventlog/[23].* ; do
        cp ${i} /tmp/${i%/*}.${i#*/} ;
done
```

The result of the above demonstration script will be a /tmp/5.eventlog.2.aap1 and /tmp/5.eventlog.3.iop files filled with the header and events dump in binary format.

Event Log resizing

The preferred method of event log sizing is to at initialization time use the

pm8001.logging_size= insmod parameter.

However, there may be a need to dynamically alter the event log size for forensic details of a series of activities. There is a requirement to preserve the existing events, within reason, when resizing is required.

A commonality between the needs of resizing the log and still preserving the existing events, and for the incremental reading of the legacy sysfs node is that the log needs to be read one member at a time. A pm8001_readlog(log header pointer, place to copy incremental read, pointer to local consumer index) routine needs to be created to manage both. A pm8001_writelog(log header pointer, new event) routine will perform the other half of the operation of resizing by placing the new entries into the log much in the same manner has the hardware would perform such an operation.

When the log is copied, the legacy handler needs to have it's private copy of the consumer index updated. Tracking the match during copy is the recommended algorithm.

The risk here is that the sequencer does not acknowledge the copy operation during resizing. Testing this may take some time.

Another risk is that there is not enough contiguous DMA memory in kernel space to resize the event log memory. The failure to acquire the memory is considered a serious fault by the kernel, so there is no gentleness in the logging. The failure can be handled by the driver by reverting, but an annoying kerneloops coredump will be logged.

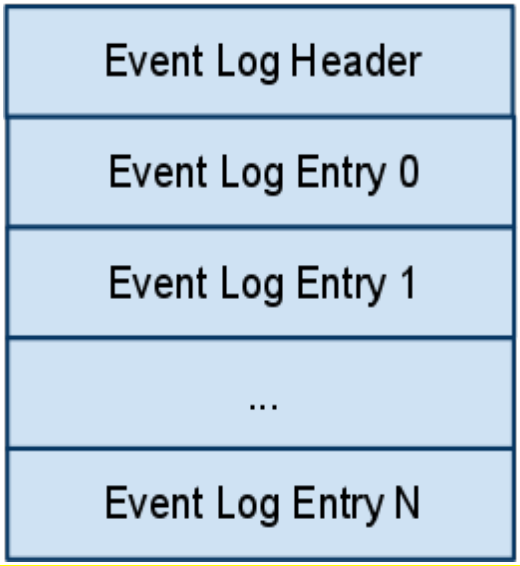
Hence the reason to acquire the memory at driver-load time before the contiguous DMA memory pool has been fragmented.

Event Log Operation

The SPC 8x6G firmware optionally supports the logging of firmware events to the host-allocated buffers. Two independent host buffers are used to log events specific to each processor (AAP1 and IOP). See Section 5.2 "MPI Configuration Table" for details on enabling this option and to set desired log severity level. To ensure all logs are in the host memory, read the "Event Log Latest Index", wait for 50 ms, re-read the "Event Log Latest Index". See Section **Event Log Header** Table for a description of the "Event Log Latest Index". If it is the same, the logs are all dumped from SPC 8x6G memory to host memory.

Figure below shows the format of the event log buffer.

Figure - Event Log Buffer Format



The event log buffer contains header information at offset 0x0 from the event log data buffer, followed by a series of fixed-sized event long entries starting from offset 0x20. The event log entries are encoded in a firmware-defined format. After the event log is retrieved, it is fed to an event log parser, which translates the event log entries to useful descriptions that can be displayed in a user-friendly way.

Event Log Header

The event log header is 32 bytes in size and the fields are described in the table below. The header size is not necessarily fixed in order to allow for future expansion.

Table - Event Log Header Format

DWord Offset	Field	Length	Description
0x00	Event Log Signature	4 Bytes	32-bit signature identifying the event log header. AAP1: 1234AAAA IOP: 5678CCCC
0x01	Event Log Entry Start Offset	4 Bytes	Specifies the byte offset of the first event log entry. The offset is calculated from the base of the event log buffer; that is, offset 0 of the event log header.
0x02	Reserved	4 Bytes	
0x03	Event Log Buffer Size	4 Bytes	Size of the event log buffer in bytes, less the header size (32 bytes). Whereas the parameter set by the HOST in the MPI configuration table is the total size including the header.

0x04	Reserved	4 Bytes	
0x05	Event Log Oldest Index	4 Bytes	Index pointer to the oldest entry in the log. Index is multiple of Event Log Entry Size.
0x06	Event Log Latest Index	4 Bytes	Index pointer to the latest entry in the log. Index is multiple of Event Log Entry Size.
0x06	Event Log Entry Size	4 Bytes	Size of each event log entry in bytes.

Event Log Latest Index is pointing to the next entry that is to be written and is in fact not the latest entry, it is actually the future entry.

Event Log Entry

Event log entries are each 32 bytes (eight 32-bit words) in the following format.

Table - Event Log Entry Format

DWord Offset	Field	Length	Description
0x00	[2:0]: Number of Words in Log Entry [27:3]: Reserved [31:28]: Event Severity	4 Bytes	Number of valid words in the entry and the event severity level.
0x01	TimeStamp Upper	4 Bytes	The upper 32 bits of the time stamp. The timer resolution is in 8-nanosecond intervals. The timer is common to both the AAP1 and the IOP.
0x02	TimeStamp Lower	4 Bytes	The lower 32 bits of the time stamp. The timer resolution is in 8-nanosecond intervals. The timer is common to both the AAP1 and the IOP.
0x03	Sequence Number	4 Bytes	Unique sequence number that is common to both the AAP1 and the IOP. Used to identify chronological order of events across processors.
0x04	Log Word 0	4 Bytes	Log specific data.
0x05	Log Word 1	4 Bytes	Log specific data.
0x06	Log Word 2	4 Bytes	Log specific data.
0x07	Log Word 3	4 Bytes	Log specific data.

Number of words in Log Entry refers to just the ‘Log Word’ fields.

MPI Configuration Table (event log focus)

Table below llists the main fields defined for the SPC 8x6G MPI configuration table. Each entry is a 32-bit DWord value. The main configuration fields defined in the Table also contains offset addresses for general status information as well as additional configuration entries defined for the IQs and OQs.

The configuration tables are located in SPC 8x6G memory space.

MPI Main Configuration Table Fields

Table - MPI Configuration Table – Main Part

DWord [Bit(s)]	Field Name	Field Symbol	Write Access	Description	Default Value
0x00	UTF8 Signature	AS	SPC 8X6G	Indicate coherent table.	Byte 0: 0x50 (hex value for UTF8 character ‘P’ Byte 1: 0x4D (hex value for UTF8 character ‘M’) Byte 2: 0x43 (hex value for UTF8 character ‘C’) Byte 3: 0x53 (hex value for UTF8 character ‘S’)
. . .					
0x17 [3:0]	MSGU Event Log Severity	MELSEV	Host	Bit field option to specify the desired severity level of the SPC 8x6G MSGU event log: 0x0: Disable logging. 0x1: Critical Error – Any firmware-detected error or top-level error interrupt that	

				disables the SPC 8x6G. Example: Firmware asserts due to an internal condition or interrupt. 0x2: Warning – Occurs when a resource is unavailable, an unexpected request occurs, or frames/ messages are dropped due to an unavailable handler. Examples: • Firmware drops a message from hardware that is unsupported or the handler is not defined. • An incoming target mode request is dropped due to resource unavailability. • The IOP receives a request from the host with an unsupported opcode or Invalid. 0x3: Notice – All transport and interconnect events and errors. Examples: • Check conditions. • All errors in command completion. • PHY down and PHY up, etc. • TMFs. • All error messages posted from hardware. 0x4: Information – Log events in the successful IO Path. Log all IO requests received and completed by the SPC 8x6G. Example: SSP_IO, SATA_IO with host tag, device ID, protocol tag, etc.	

				0x5: Debugging – This is a special level meant to debug consistently reproducible issues by adding more logs while debugging. This level is currently for internal use. 0x6-0xF: Reserved	
0x17 [31:4]	Reserved				
. . .					
0x1B [3:0]	IOP Event Log Severity	IELSEV	Host	Bit field option to specify the desired severity level of the SPC 8x6G IOP event log: 0x0: Disable logging. 0x1: Critical Error – Any firmware-detected error or top-level error interrupt that disables the SPC 8x6G. Example: Firmware asserts due to an internal condition or interrupt. 0x2: Warning – Occurs when a resource is unavailable, an unexpected request occurs, or frames/ messages are dropped due to an unavailable handler. Examples: • Firmware drops a message from hardware that is unsupported or the handler is not defined. • An incoming target mode request is dropped due to resource unavailability. • The IOP receives a request from the host with an unsupported opcode or Invalid.	

				0x3: Notice – All transport and interconnect events and errors. Examples: • Check conditions. • All errors in command completion. • PHY down and PHY up, etc. • TMFs. • All error messages posted from hardware. 0x4: Information – Log events in the successful IO Path. Log all IO requests received and completed by the SPC 8x6G. Example: SSP_IO, SATA_IO with host tag, device ID, protocol tag, etc. 0x5: Debugging – This is a special level meant to debug consistently reproducible issues by adding more logs while debugging. This level is currently for internal use. 0x6-x0F: Reserved.	
0x1B [31:4]	Reserved				
. . .					

MPI Configuration Table

Placed at .../forensic/6.mpi_config.

Script to retrieve the complete dump:

```
# cat forensic/6.mpi_config > /tmp/6.mpi_config
```

Please provide the MPI configuration table in an UTF8 hexadecimal format (firmware

revision information highlighted). The MPI configuration information is stored in configuration table, a memory region mapped to the PCIe memory space and accessible by the host. See [1], Section 5, “Message Passing Interface (MPI) Configuration”.

```
[0000] 53434d50 00000001 01071003 00001000 04000000 0c454040 0000008c 000000f0
[0020] 000008f0 00000020 00000000 09060300 00000000 0a070401 00000000 0a070401
[0040] 00000000 0a070401 00000000 00000000 00000003 799f0060 00380000 00000004
[0060] 00000003 79d70060 00380000 00000004 00000000 0000001c 00003fc0 0000401c
[0080] 00003fc0 0000000e 000011f0
```

MPI General Status Table (GST)

Placed at .../forensic/7.gst.

Script to retrieve the complete dump:

```
# cat forensic/7.gst > /tmp/7.gst
```

Please provide the MPI status table in an UTF8 hexadecimal format. See [1], Section 5.2.2, “MPI General Status Table Fields”. All fields in the GST are host read-only access.

```
[0000] 91010000 00000000 00000000 85000000 7B000000 00000000 01000000 03000000
[0020] 01000000 01000000 01000000 01000000 01000000 01000000 F9FF0600 00000000
[0040] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[0060] 00000000
```

MPI Queue Configuration Tables

Placed at .../forensic/8.mpi_queue

Script to retrieve the complete dump:

```
# cat forensic/8.mpi_queue > /tmp/8.mpi_queue
```

We are not separating the IBQ from the OBQ, both will be present in the output.

Provide a file or files with the lists of the configuration parameters for both the IQs and OQs. Show all the inbound and outbound queue configuration tables in an UTF8 hexadecimal format. This example shows one IQ and one OQ configuration tables. See [1], Section 5.2.3, “MPI Inbound Queue Configuration Table Fields” and Section 5.2.4, “MPI Outbound Queue Configuration Table Fields”.

MPI IBQ 0 Configuration Table

```
[0000] 01044000 00000000 0090BE04 00000000 8043E104 24000000 00D80000 00000000
```

MPI OBQ 0 Configuration Table

[0000] 01044040 00000000 4097BF04 00000000 8C43E104 24000000 00E00000 00000000
[0020] 00000000

Debug Forensics

The forensic data needs to be collected and bundled into an archive. We must make sure that the driver has the debugfs enabled, and that the OS has debugfs properly mounted to perform the operation. A fundamental limitation of debugfs for tar or cpio archiving is that all the files report zero size in the directory, yet do in fact have a dynamic content. The results is that tar (or cpio) will not transfer any content into the archive. The following script demonstrates an example of the collection procedure; first the data is transferred to a temporary filesystem location, so that the content can be archived::

```
# [ -d /sys/kernel/debug/pm8001 ] || mount -tdebugfs none /sys/kernel/debug
# ( cd /sys/kernel/debug && find pm8001 -type f ) | while read i ; do \
        mkdir -p /tmp/${i%/*} && \
        cp /sys/kernel/debug/${i} /tmp/${i} ; \
  done
# tar -czf /tmp/pm8001.${$}.tgz /tmp/pm8001/
# lspci -xxxx -vvv  -d 11F8:8001 >/tmp/lspci.${$}.txt
# tar -Azf /tmp/pm8001.${$}.tgz /tmp/lspci.${$}.txt
```

Note: Version information missing (but embedded in the raw data in the above logs).

Driver Build

Example build that ensures that the feature is enabled:

```
# make -C /lib/modules/linux-2.6.32-71.18.2.el6.lustre.x86_64 \
        SUBDIRS=`pwd` \
        MODFLAGS='-DMODULE -D_CONFIG_SCSI_PM8001_DEBUG_FS'
```

We need to ensure that this is incorporated into the spec-file as incorporated into our release or consider defining _CONFIG_SCSI_PM8001_DEBUG_FS internally and consider that this is a piece that would not be propagated to the community.

Architectural Challenges & Risks

- few examples of debugfs coding to leverage reliability, all new code.
- the perennial battle between binary and UTF8 representation
- Lack of control over CONFIG_SCSI_PM8001_DEBUG_FS
- The requirement that debugfs filesystem order is preserved in POSIX Bourne Shell expansion so that ‘ls *’ produces the same effective results as (A single layer) ‘find’ might not be possible. This is done so that there is no need for an intervening sort, and for any archive tools (tar, cpio, zip etc) to pick up files in the expected order. Do your best.
- Ensuring that the source changes work on CentOS 5.4 to latest, RHEL6 to latest, Scientific Linux to latest, Manufacturing kernels (2.6.26-2-686-bigmem-kci-1.0) and on kernel.org latest.

Future Directions

- submit to kernel.org
- Add atomic handling (see unlock comment in queue section as an example).