

The Linux driver for the Rome PCI-AER board

[Adrian M. Whatley, Institute of Neuroinformatics, University & ETH Zurich](#)

Specification version [2.16 \(4.1.2012\)](#) ~~2.15 (22.12.2011)~~ for driver versions ≥ 2.454

Previous specification versions:

2.15	22.12.2011
2.14	28.9.2010
2.13	5.7.2010
2.12	1.4.2010
2.11	25.3.2010
2.10	4.2.2009
2.9	21.10.2008
2.8	20.3.2007
2.7	17.1.2006
2.6	12.7.2005
2.5	25.5.2005
2.4	15.4.2004
2.3	19.1.2004
2.2	3.7.2003
2.1	26.5.2003
2.0	9.5.2003
unnumbered	9.8.2000

General

The Linux 2.6.x driver for the PCI-AER board [1] is contained in an installable module called *pciaer.ko*. It supports (subject to the alteration of a compile time constant) up to four PCI-AER boards in one machine. The logically separate functions of the [Mapper](#), [Monitor](#) and [Sequencer](#) are supported by three minor devices for each board, together with an additional minor device reserved for accessing the [PCI interface configuration NVRAM](#) [2]. The separation of Mapper, Monitor, and Sequencer has the advantage of giving the desired degree of granularity of control and leaves open the possibility of configuring the Mapper via a human-readable read/write call interface in addition to via its ioctl interface, and in addition to the read and write calls necessary to support the Monitor and Sequencer.

The minor device for accessing the PCI interface configuration NVRAM is intended only as a convenience for the hardware developer, and is not intended for use by the end user.

In each of the descriptions of the minor devices that follow, there are entries for most of the possible file operations which are relevant to character device drivers. The exceptions are `fasync`, `aio_read`, `aio_write`, and `aio_fsync` which are not supported since asynchronous operations are not supported, `flush` and `check_flags`. All ioctl calls are handled via `unlocked_ioctl`.

Unless otherwise described below, the state of the hardware is preserved across `open` and `close` calls on the Mapper, Monitor and Sequencer devices, such that connected AER systems continue to function without interruption.

A header file *pciaer.h* contains definitions of the constants (e.g. ioctl numbers) and structures described in this document.

Inserting the Driver into the Kernel

Both dynamic and static major number allocation will be supported. In the former, default case, the driver can be inserted using *insmod* or *modprobe* without specifying any load-time parameters. The kernel will pick a free major number and use that. The user can then determine which number has been selected by examining `/proc/devices` and can create the corresponding device nodes appropriately. This can of course be automated, see for example the script in [3]. Alternatively, the device nodes can be created first, and the major number chosen by the user can be supplied to the driver as a load-time parameter on the *insmod* command line or in the *modprobe.conf* file, e.g.:

```
insmod pciaer.o major=60
```

The driver will only accept major numbers supplied in this way which are in one of the ranges listed in [4] as allocated for local/experimental use, namely 60-63, 120-127 and 240-254, all inclusive.

The Device Filesystem *devfs*, in which a device driver causes entry points in */dev* to be created and removed automatically at driver initialization and removal times respectively, will not be supported.

Users should use the `make install` target to install the driver into the correct kernel module *extra* directory and then run `/sbin/modprobe pciaer` to insert the driver into the kernel and `/sbin/modprobe -r pciaer` to remove it. The major number will then be supplied on the command line automatically according to the entry made in the `modprobe.conf.local` file during installation.

The module is marked as being GPL'd, but SuSE kernels may still complain about being 'tainted' when the module is inserted because the module is not supported by SuSE.

Module parameters

~~In addition to the above mentioned module parameter called `major`, there are further integer~~ The available module parameters are described in the following table. ~~called `allow_mmap_write`, `nvr_only` and `no_sample_random`, all of which default to zero.~~

<u>Name</u>	<u>Type</u>	<u>Default</u>	<u>Function</u>
<code>major</code>	<code>uint</code>	<code>0</code>	<u>If non-zero, this sets the major number used by the driver, see above.</u>
<code>allow_mmap_write</code>	<code>bool</code>	<code>false</code>	<u>If true, permit programs to use <code>mprotect</code> to gain write access to the Mapper SRAM when it has previously been <code>mmap</code>'ed read-only. This should only be used for hardware debugging purposes, as the integrity of the mapping tables can no longer be guaranteed.</u>
<code>nvr_only</code>	<code>bool</code>	<code>false</code>	<u>If true, the driver only attempts to make the PCI interface configuration NVRAM minor device usable. This is useful on boards which have not yet been configured, or which have lost their configuration such that the driver cannot be loaded normally.</u>
<code>no_sample_random</code>	<code>bool</code>	<code>false</code>	<u>If true, the timing of interrupts from the PCI-AER card will not be allowed to contribute to the entropy pool used to generate random numbers. This might be important if very regular address-event senders and receivers are in use (e.g. during test) but it is still important that random-number generation be of high quality.</u>
<code>abort_init_at</code>	<code>uint</code>	<code>0</code>	<u>Setting this non-zero can be used to abort the driver initialization process for debugging purposes at the point defined by an instance in the source of the <code>ABORT_INIT_POINT</code> macro. Only available if <code>CONFIG_PCIAER_DEBUG</code> is defined.</u>

~~If `allow_mmap_write` is set non-zero then the driver will permit programs to use `mprotect` to gain write access to the Mapper SRAM when it has previously been `mmap`'ed read-only. This should only be used for hardware debugging purposes, as the integrity of the mapping tables can no longer be guaranteed.~~

~~If `nvr_only` is set non-zero then the driver will load with only the PCI interface configuration NVRAM minor device usable.~~

~~If `no_sample_random` is set non-zero then the timing of interrupts from the PCI-AER card will not be allowed to contribute to the entropy pool used to generate random numbers. This might be important if very regular address-event senders and receivers are in use (e.g. during test) but it is still important that random-number generation be of high quality.~~

~~If the driver is compiled with the C pre-processor symbol `PCIAER_DEBUG` defined, then another integer module parameter called `abort_init_at` is available. Setting this non-zero can be used to abort the driver initialization process for debugging purposes at the point defined by an instance in the source of the `ABORT_INIT_POINT` macro.~~

When the module is inserted into the kernel, the module parameters are readable from files in `/sys/module/pciaer/parameters/`.

Minor device numbers and device node naming

Minor devices are numbered and named as follows:

0 = /dev/aernvr0	First PCI-AER board config NVRAM
1 = /dev/aermap0	First PCI-AER board Mapper
2 = /dev/aermon0	First PCI-AER board Monitor
3 = /dev/aerseq0	First PCI-AER board Sequencer
4 = /dev/aernvr1	Second PCI-AER board config NVRAM
5 = /dev/aermap1	Second PCI-AER board Mapper
6 = /dev/aermon1	Second PCI-AER board Monitor
7 = /dev/aerseq1	Second PCI-AER board Sequencer
8 = /dev/aernvr2	Third PCI-AER board config NVRAM
9 = /dev/aermap2	Third PCI-AER board Mapper
10 = /dev/aermon2	Third PCI-AER board Monitor
11 = /dev/aerseq2	Third PCI-AER board Sequencer
12 = /dev/aernvr3	Fourth PCI-AER board config NVRAM
13 = /dev/aermap3	Fourth PCI-AER board Mapper
14 = /dev/aermon3	Fourth PCI-AER board Monitor
15 = /dev/aerseq3	Fourth PCI-AER board Sequencer

The permissions on the `aernvr*` device nodes should be set such that only root can write to the PCI interface configuration NVRAM, otherwise it might become possible for non-root users to write boot ROM code into the NVRAM and take complete control of the PC at next boot. (Writing to the NVRAM is however further protected by checking for the capability `CAP_SYS_ADMIN`.)

Monitor minor device

This minor device is used to read from the monitor FIFO and control other aspects of the hardware related to the Monitor.

open

The Monitor device can be opened as many times as desired provided that it is opened with the same effective user ID (euid) and that neither Sequencer nor Mapper have been opened with a different euid. Attempting to open it with a different euid will fail, with `errno` being set to `EBUSY`. Otherwise no special action is taken on opening the device, since the Monitor is always enabled to prevent sending devices (chips) from hanging waiting for a request to be acknowledged. Note that when the Monitor device is opened, the Monitor FIFO may contain stale, possibly very stale data. Applications must call the `PCIAER_IOC_RST_MON_FIFO` ioctl call if they want to be sure of reading recent data. The `open` may block (even when `O_NONBLOCK` is specified), but only in low memory situations.

release

On release, no special action is taken, since the Monitor is always enabled to prevent sending devices (chips) from hanging waiting for a request to be acknowledged. The Monitor FIFO will continue to be filled by incoming address-events.

llseek

Seeking makes no sense for the Monitor, so all attempts to seek on the device are failed, and `errno` is set to `ESPIPE`.

read

The 32-bit data words read from are formatted as follows (see Table 1 in [1]):

Value of (<i>data_word & MONITOR_DWORD_TYPE_MASK</i>)	Meaning of (<i>data_word & MONITOR_DWORD_DATA_MASK</i>)
<code>MONITOR_DWORD_AER_ADDR</code>	AE address value
<code>MONITOR_DWORD_TIME_HI</code>	High order word of time at which AE occurred
<code>MONITOR_DWORD_TIME_LO</code>	Low order word of time at which AE occurred
<code>MONITOR_DWORD_ERROR</code>	Error code

Both blocking and non-blocking read will be supported. If the Monitor FIFO is empty, in the blocking case `read` will block whereas in the non-blocking case the read will fail with `errno` set to `EAGAIN`. Otherwise, if the Monitor FIFO is not empty, as many `DWORD`s as possible will be read from the FIFO and placed into the user's buffer (i.e. until the FIFO becomes empty, or the end of the user's buffer is reached). In order to prevent possible confusion about the meaning of the next byte read which could arise after reading a number of bytes not divisible by four, the driver will never return a non-integral number of `DWORD`s. The return value is the number of *bytes* read. There are no circumstances in which `0` (indicating end of file) can be returned, unless the caller specified that zero bytes should be read. If greater than zero but less than four bytes are requested, the read will fail with `errno` set to `EINVAL`.

For efficiency, applications should be prepared to read the preferred blocksize of data from the Monitor. This can be determined by calling `fstat(2)` on the Monitor and examining the `st_blksize` member of the returned `stat` structure. This field gives the "preferred" blocksize (in bytes) for reads, which is currently half of the FIFO depth in `DWORD`s multiplied by the size of a `DWORD`. Note that when time labels are enabled, each event requires three FIFO words, so a FIFO with a depth of 64K could hold a maximum of 21845 complete time-stamped events.

Since the Monitor is a non-seekable device, the `pread` system call is failed with `errno` being set to `ESPIPE`.

write

Writing to the Monitor is meaningless and `write` will not be implemented. The kernel will return `-EINVAL` to the calling program.

poll

Whenever the Monitor FIFO is non-empty, `poll` will indicate that the device can be read without blocking. It will always indicate that the device can be written without blocking, since `write` will always immediately return an error.

mmap

There is nothing that can sensibly be mapped for the Monitor, so `mmap` will not be implemented. The kernel will return `-ENODEV` to the calling program.

fsync

Will not be implemented since there can be no pending data to flush on the read-only Monitor device. The kernel will return `-EINVAL` to the calling program.

unlocked_ioctl

The following ioctls are defined on the Monitor minor device:

FIONREAD

This standard ioctl (see `man tty_ioctl`) sets the `int` pointed to by its argument to the number of *bytes* available for reading.

PCIAER_IOC_GET_VERSION_INFO
PCIAER_IOC_SET_CNTR_PERIOD
PCIAER_IOC_GET_CNTR_PERIOD
PCIAER_IOC_RST_CNTR_GET_TIME
PCIAER_IOC_GET_CNTR_LAST_RST_TIME
PCIAER_IOC_SET_ARB_CONFIG
PCIAER_IOC_GET_ARB_CONFIG

PCIAER_IOC_RST_FIFO
PCIAER_IOC_GET_PCI_INFO *(deprecated)*
PCIAER_IOC_GET_STATS *(deprecated)*
PCIAER_IOC_RST_STATS *(deprecated)*

These ioctls are described in the Common Ioctls section [below](#).

PCIAER_IOC_SET_MON_CH_SEL
PCIAER_IOC_GET_MON_CH_SEL

These two ioctls deal with which channels are actually monitored. The `...SET...` ioctl uses its argument directly to specify a 4-bit mask containing a `1` for each channel which should be monitored, and a `0` for each channel which should not be monitored. The `...GET...` ioctl fills the `int` pointed to by its argument with such a bit mask according to the current status. The `...SET...` ioctl requires write access to the device; the `...GET...` ioctl requires read access.

PCIAER_IOC_SET_MON_TIME_LBL_FLAG
PCIAER_IOC_GET_MON_TIME_LBL_FLAG

These two ioctls deal with the Monitor's Time Label flag. If the argument of the `...SET...` ioctl is zero, time labels will be disabled, otherwise they will be enabled. The `...GET...` ioctl sets the `int` pointed to by its argument to `0` or `1` according to the current status of the flag. The `...SET...` ioctl requires write access to the device; the `...GET...` ioctl requires read access.

PCIAER_IOC_GET_MON_FIFO_FLAGS

Fills the `int` pointed to by its argument with a status word containing a positive logic copy of the Monitor FIFO flag bits, i.e. some combination of the following defined flag bits: `PCIAER_IOC_MON_EMPTY`, `PCIAER_IOC_MON_HALF_FULL` and `PCIAER_IOC_MON_FULL`, together with some other reserved bits. This call can be used to determine whether a subsequent read call might be able to read a half or a full FIFO's worth of data. Just how much that is depends on the depth of the FIFO. This ioctl requires read access to the device.

Sequencer minor device

This minor device is used to write to the Sequencer FIFO and control other aspects of the hardware related to the Sequencer.

open

The Sequencer device can be opened as many times as desired provided that it is opened with the same effective user ID (euid) and that neither Monitor nor Mapper have been opened with a different euid. Attempting to open it with a different euid will fail, with `errno` being set to `EBUSY`. The Sequencer hardware is not enabled on open, but only at the time of first write (to enable initial configuration via `ioctl`s).

release

If the Sequencer FIFO is not empty release will block until the FIFO is empty or a signal is received. The Sequencer hardware is then disabled.

llseek

Seeking makes no sense for the Sequencer, so all attempts to seek on the device are failed, and `errno` is set to `ESPIPE`.

read

Reading from the Sequencer is meaningless and `read` will not be implemented. The kernel will return `-EINVAL` to the calling program.

write

Data words to be written to the Sequencer FIFO should be formatted as follows (see Table 2 in [1]):

Value of (<i>data_word</i> & SEQUENCER_DWORD_TYPE_MASK)	Meaning of (<i>data_word</i> & SEQUENCER_DWORD_DATA_MASK)
SEQUENCER_DWORD_END_SEQUENCE	None
SEQUENCER_DWORD_AER_ADDR	AE address value
SEQUENCER_DWORD_DELAY	Relative delay in units of the current AER Clock Period
SEQUENCER_DWORD_WAIT_TIME	Absolute value of counter to wait for

The buffer to be written should be an integral number of DWORDs long, otherwise the write will fail with `errno` set to `EINVAL`. This unconventional policy will be implemented in order to prevent the possible confusion in the meaning of the next byte to be written which could arise after writing a number of bytes not divisible by four.

On first writing to the device, the Sequencer hardware is enabled.

Both blocking and non-blocking write will be supported. If the Sequencer FIFO is full, in the blocking case `write` will block whereas in the non-blocking case the write will fail with `errno` set to `EAGAIN`. There are no circumstances in which the error `ENOSPC` will be returned. If the Sequencer FIFO is not full, as many words as possible will be written to the FIFO from the user's buffer (i.e. until the FIFO becomes full or the end of the user's buffer is reached). If the `O_SYNC` file flag is set and `O_NONBLOCK` is not set, the write call will not return until all of the words in the user's buffer have been written to the FIFO. The return value is the number of *bytes* written.

For efficiency, applications should write the preferred blocksize of data to the Sequencer. This can be determined by calling `fstat` (2) on the Sequencer and examining the `st_blksize` member of the returned `stat` structure. This field gives the "preferred" blocksize (in bytes) for writes, which is currently half of the FIFO depth in DWORDs multiplied by the size of a DWORD. Note that for the Sequencer FIFO each event *typically* requires two words, a delay and an address, so a 64K deep Sequencer FIFO might hold only 32K events, but might hold slightly more if not every event requires a delay, or slightly less if some back to back delays are required between events.

Since the Sequencer is a non-seekable device, the `pwrite` system call is failed with `errno` being set to `ESPIPE`.

poll

Whenever the Sequencer FIFO is non-full, `poll` will indicate that the device can be written to without blocking. It will always indicate that the device can be read without blocking, since `read` will always immediately return an error.

mmap

There is nothing that can sensibly be mapped for the Sequencer, so `mmap` will not be implemented. The kernel will return `-ENODEV` to the calling program.

fsync

If the Sequencer FIFO is not empty, `fsync` will block until it is empty.

unlocked_ioctl

The following `ioctl`s are defined on the Sequencer minor device:

```
PCIAER_IOC_GET_VERSION_INFO  
PCIAER_IOC_SET_CNTR_PERIOD  
PCIAER_IOC_GET_CNTR_PERIOD  
PCIAER_IOC_RST_CNTR_GET_TIME  
PCIAER_IOC_GET_CNTR_LAST_RST_TIME  
PCIAER_IOC_SET_SEQ_ARB_CHANNEL  
PCIAER_IOC_GET_SEQ_ARB_CHANNEL  
  
PCIAER_IOC_RST_FIFO  
PCIAER_IOC_GET_PCI_INFO (deprecated)  
PCIAER_IOC_GET_STATS (deprecated)  
PCIAER_IOC_RST_STATS (deprecated)
```

These `ioctl`s are described in the Common `Ioctl`s section [below](#).

PCIAER_IOC_GET_SEQ_FIFO_FLAGS

Fills the `int` pointed to by its argument with a positive logic copy of the Sequencer FIFO flag bits, i.e. some combination of the following defined flag bits: `PCIAER_IOC_SEQ_EMPTY`, `PCIAER_IOC_SEQ_HALF_FULL` and `PCIAER_IOC_SEQ_FULL`. This `ioctl` requires read access to the device.

Mapper minor device

This minor device is used to read and write the Mapper SRAM and to control other aspects of the hardware related to the Mapper. The primary means of access to the Mapper SRAM is intended to be via an `ioctl` interface which provides various commands for manipulating lists of address-event mappings. The Mapper SRAM can also be read by using `mmap` to map it into user space, but reading from this memory will only produce valid results while mapper output is disabled and the mapper is not busy (see the `PCIAER_IOC_*_MAP_OUT*` group of `ioctls`). This is intended primarily for use in debugging, and direct write access is not permitted in order to guarantee the integrity of the mapping tables. The file operations `read` and `write` are not implemented, but could in the future support a basic human-readable command interface.

open

The Mapper device can be opened as many times as desired provided that it is opened with the same effective user ID (`eid`) and that neither Monitor nor Sequencer have been opened with a different `eid`. Attempting to open it with a different `eid` will fail, with `errno` being set to `EBUSY`. Otherwise no special action is taken on opening the device, as the hardware is always enabled by default. The `open` will never block.

release

On release, no special action is taken, unless mapper output has been disabled using `PCIAER_IOC_DIS_MAP_OUT`, in which case output is re-enabled.

llseek

Seeking makes no sense for the Mapper, so all attempts to seek on the device are failed, and `errno` is set to `ESPIPE`.

read

Will not be implemented so the kernel will return `-EINVAL` to the calling program.

write

Will not be implemented so the kernel will return `-EINVAL` to the calling program.

poll

Will not be implemented.

mmap

Will map the hardware SRAM to user space with read-only access. If the module parameter `allow_mmap_write` is set non-zero then the driver will permit programs to use `mprotect` to gain write access to the Mapper SRAM when it has previously been `mmap`'ped read-only. This should only be used for hardware debugging purposes, as the integrity of the mapping tables can no longer be guaranteed.

fsync

Will not be implemented so the kernel will return `-EINVAL` to the calling program.

unlocked_ioctl

PCIAER_IOC_GET_VERSION_INFO
PCIAER_IOC_SET_SEQ_ARB_CHANNEL
PCIAER_IOC_GET_SEQ_ARB_CHANNEL
PCIAER_IOC_SET_ARB_CONFIG
PCIAER_IOC_GET_ARB_CONFIG

PCIAER_IOC_RST_FIFO
PCIAER_IOC_GET_PCI_INFO *(deprecated)*
PCIAER_IOC_GET_STATS *(deprecated)*
PCIAER_IOC_RST_STATS *(deprecated)*

These ioctls are described in the Common Ioctls section [below](#).

PCIAER_IOC_SET_MAP_CH_SEL
PCIAER_IOC_GET_MAP_CH_SEL

These two ioctls deal with which channels' input is actually processed by the mapper. The ...SET... ioctl uses its argument directly to specify a 4-bit mask containing a 1 for each input channel which should be processed by the mapper, and a 0 for each channel which should be ignored by the mapper. The ...GET... ioctl fills the `int` pointed to by its argument with such a bit mask according to the current status. The ...SET... ioctl requires write access to the device; the ...GET... ioctl requires read access.

PCIAER_IOC_SET_MAP_MAPPING
PCIAER_IOC_GET_MAP_MAPPING_COUNT
PCIAER_IOC_GET_MAP_MAPPING
PCIAER_IOC_MAP_ADD_TO_MAPPING
PCIAER_IOC_MAP_DEL_FROM_MAPPING

These five ioctls all use their argument to point to a `pciaer_mapping` structure

```
struct pciaer_mapping {
    unsigned short source_ae;
    unsigned short dest_count;
    unsigned short *dest_ae;
};
```

where `dest_ae`, when valid, points to an array of at least `dest_count` unsigned shorts, and all five cause mapping output to be suspended for the duration of the call.

In the ...SET_MAP_MAPPING case, all fields must be valid on entry, and the call establishes a new mapping or replaces the existing mapping from the given `source_ae` to the given list of `dest_ae`, or if `dest_count` is 0, deletes any existing mapping for the given `source_ae`. This ioctl requires write access to the device.

In the ...GET_MAP_MAPPING_COUNT case, only the `source_ae` field must be valid on entry and the driver fills in the `dest_count` field. This ioctl requires read access to the device.

In the ...GET_MAP_MAPPING case, the driver fills the `dest_ae` array with the list of destinations for the given `source_ae` and replaces `dest_count` with the actual number of destinations copied to `dest_ae`. If on entry `dest_count` is less than the number of destination addresses there currently are for the given `source_ae`, `EINVAL` is returned. This ioctl requires read access to the device.

In the ...ADD_TO_MAPPING and ...DEL_FROM_MAPPING cases, all fields must be valid on entry, and the given list of `dest_ae` will be added or deleted respectively from any existing list for the given `source_ae`. These ioctls require write access to the device.

Both `PCIAER_IOC_SET_MAP_MAPPING` and `PCIAER_IOC_MAP_ADD_TO_MAPPING` may fail returning `ENOSPC` due to there being insufficient free contiguous Mapper memory available, in which case `PCIAER_IOC_MAP_COMPACT` may help.

In all cases, the address-event `FFFFh` is reserved, and attempting to use this will result in the error `EINVAL`.

PCIAER_IOC_FIND_NEXT_MAP_MAPPING

Replaces the `int` pointed to by its argument with the next source address-event for which a mapping exists after the one specified, or -1 if there are no more. To find the first mapped address, supply -1 in the `int` pointed to by the argument. This ioctl causes mapping output to be suspended for the duration of the call and requires read access to the device.

PCIAER_IOC_GET_MAP_MAPPED_MAP

Fills the 8K of memory pointed to by its argument with a bit vector in which a 0 represents a source address-event for which a mapping does not exist and a 1 represents one for which a mapping does exist. This ioctl causes mapping output to be suspended for the duration of the call and requires read access to the device.

PCIAER_IOC_MAP_CLEAR

Clears the Mapper's memory to a state in which no address-events are mapped. No argument is used. This ioctl causes mapping output to be suspended for the duration of the call and requires write access to the device.

PCIAER_IOC_MAP_COMPACT

Forces maximal compaction of the Mapper's address table. No argument is used. This ioctl causes mapping output to be suspended for the duration of the call and requires write access to the device.

PCIAER_IOC_GET_MAP_FREE_SPACE

Fills the unsigned `int` pointed to by its argument with the number of free words in the mapper SRAM. This gives an indication of how many more destination addresses could be stored, but the free space may be fragmented and need compaction before it is used. This ioctl requires read access to the device.

PCIAER_IOC_GET_MAP_FIFO_FLAGS

Fills the `int` pointed to by its argument with a positive logic copy of the Mapper FIFO flag bits, i.e. some combination of the following defined flag bits: `PCIAER_IOC_MAP_EMPTY` and `PCIAER_IOC_MAP_FULL`. This ioctl requires read access to the device.

PCIAER_IOC_ENABLE_MAP_OUT

PCIAER_IOC_DISABLE_MAP_OUT

PCIAER_IOC_GET_MAP_OUT_STATE

These three ioctls deal with whether output from the mapper is or is not enabled. By default, mapper output is enabled, but reading from the mapper's hardware SRAM via the memory mapped into user space using `mmap` will only produce valid results while mapper output is disabled and the mapper is not busy. Therefore, these ioctls are provided to allow a user program to temporarily disable mapper output and wait for the mapper to become idle before it inspects the RAM and to enable mapper output again afterwards. The `...ENABLE...` ioctl uses no argument. The `...DISABLE...` ioctl uses its argument directly; if the argument is 0, the call returns immediately after disabling the mapper output, and it is the user's responsibility to determine when the mapper becomes idle; if the argument is non-zero, the call will not return until the mapper has become idle and the results of reading from the RAM will be meaningful. The `...GET...` ioctl fills the `int` pointed to by its argument with a status word containing a combination of the bits `PCIAER_IOC_MAP_OUT_ENABLED` and `PCIAER_IOC_MAP_OUT_BUSY`. Both bits must be zero before accesses to the mapper RAM is meaningful. The `...ENABLE...` and `...DISABLE...` ioctls require write access to the device; the `...GET...` ioctl requires read access.

PCIAER_IOC_SET_MAP_OUT_CONFIG

PCIAER_IOC_GET_MAP_OUT_CONFIG

These two ioctls deal with the configuration of the Mapper output. The `...SET...` ioctl uses its argument directly to specify one of the values `PCIAER_IOC_MAP_OUT_PASS_THRU`, `PCIAER_IOC_MAP_OUT_1_TO_1` or `PCIAER_IOC_MAP_OUT_1_TO_MANY`. The `...GET...` ioctl fills the `int` pointed to by its argument with one of these values according to the current status. The `...SET...` ioctl requires write access to the device; the `...GET...` ioctl requires read access.

PCIAER_IOC_SET_MAP_DEMUX_CONFIG
PCIAER_IOC_GET_MAP_DEMUX_CONFIG

These two ioctls deal with the configuration of the AER demultiplexer on the Mapper output. The ...SET... ioctl uses its argument directly to specify one of the values PCIAER_IOC_MAP_DEMUX_0_16, PCIAER_IOC_MAP_DEMUX_1_15 or PCIAER_IOC_MAP_DEMUX_2_14. The names of these constants reflect the way the 16 bits are split between channel number and actual AE bits. The ...GET... ioctl fills the `int` pointed to by its argument with one of these values according to the current status. The ...SET... ioctl requires write access to the device; the ...GET... ioctl requires read access.

PCIAER_IOC_SET_MAP_AER_PROTOCOL
PCIAER_IOC_GET_MAP_AER_PROTOCOL

These two ioctls deal with the type of AER protocol used on the Mapper output. The ...SET... ioctl uses its argument directly to specify one of the values PCIAER_IOC_MAP_AER_P2P or PCIAER_IOC_MAP_AER_SCX indicating the classic point to point, four phase handshake protocol or the shared bus, data only driven on acknowledge protocol respectively. The ...GET... ioctl fills the `int` pointed to by its argument with one of these values according to the current status. The ...SET... ioctl requires write access to the device; the ...GET... ioctl requires read access.

PCI interface configuration NVRAM minor device

This minor device can be used to access the non-volatile memory associated with the [S5920 PCI interface chip](#) in which PCI configuration data (and optionally an expansion BIOS) are stored. This is intended to assist the hardware developer with programming and debugging this device. Due to the adverse effects that reprogramming this memory could have (including making the host unbootable without removing the PCIAER board!), only processes with the capability `CAP_SYS_ADMIN` will be able to write to this minor device. The driver will also not allow writes (other than of the required values) to certain particularly sensitive bytes. Reading the NVRAM can be performed via `read`, which will cause NVRAM to be read and cached in RAM by the driver. Writes will normally go only to the cache and will not be written to the physical NVRAM unless the `O_SYNC` flag is specified, or until the device is released or `fsync` is called. ~~Reading or writing single bytes can also be performed via a deprecated `ioctl` interface.~~ Non-blocking operation is not supported.

open

On `open`, the byte at offset 2 within the NVRAM will be read to obtain an indication of its length. (There is no way to determine the actual size of the NVRAM device fitted to a board.) This means that `open` will block. If the length byte does not contain a valid value, the NVRAM will be assumed to be the minimum size possible, i.e. 128 bytes. To access larger NVRAMs, the length byte should be written to, and the device closed and re-opened. If an attempt is made to open the device specifying `O_NONBLOCK`, `open` will return `-EINVAL`.

release

On `release`, any dirty cached bytes will be written back to the NVRAM. This may take a long time (up to in excess of 20s for the whole of a 2K NVRAM) since the NVRAM enters a “shut down” state lasting 5 – 10ms between each byte written.

lseek

Will function in the conventional way, where the position within the file is to be understood as a position within the cached image of the NVRAM. A seek to position 0 with respect to `SEEK_END` can be used to determine the driver's current notion of the size of the NVRAM.

read

Only blocking read will be supported. As many bytes from the current position as are already cached will be read and returned immediately, if any. If the first byte at the current file position is not yet cached, `read` will block while that byte is read. A value of 0 (indicating end of file) can be returned if an attempt is made to read beyond the driver's current notion of the size of the NVRAM.

write

Writing will not normally block, since it will usually write to the driver's cached copy of NVRAM, but if the `O_SYNC` flag is turned on, `write` will block while the byte or bytes is or are written to the physical NVRAM. This may take a long time (up to in excess of 20s for the whole of a 2K NVRAM) since the NVRAM enters a “shut down” state lasting 5 – 10ms between each byte written. A value of 0 (indicating end of file) can be returned if an attempt is made to write beyond the end of NVRAM. Attempts to write to bytes 40h, 41h, 42h, 43h, 50h, 51h, 52h or 53h will be ignored unless the value to be written is the required value for that location (see Table 1 on page 2-74 of [2]) or in the case of bytes 42h and 43h is the value 20h or 59h respectively. (These latter values represent the device ID 5920 which, if changed, would prevent the driver from recognising the card after a reboot.)

poll

The `poll` method will always indicate by returning with the `POLLIN` bit set that the device can be read from, even if the read would block, since only the act of attempting the read can cause new data to become available. Whether or not `poll` indicates by returning with the `POLLOUT` bit set that the device can be written to without blocking depends only on whether the `O_SYNC` flag is set, since if it is not, `write` always writes only to the cache, whereas if `O_SYNC` is set, `write` will block while writing to the hardware. At end of file, `POLLHUP` will be returned.

fsync

Any dirty cached bytes will be written back to the NVRAM. This may block for a long time (up to in excess of 20s for a 2K NVRAM) since the NVRAM enters a “shut down” state lasting 5 – 10ms between each byte written.

unlocked_ioctl

PCIAER_IOC_GET_VERSION_INFO

PCIAER_IOC_GET_PCI_INFO (*deprecated*)

These ioctls are described in the Common Ioctls section [below](#).

PCIAER_IOC_READ_NVR_BYTE (deprecated)

PCIAER_IOC_WRITE_NVR_BYTE (deprecated)

~~These two deprecated ioctls deal with reading and writing single bytes of NVRAM data. Both use a `pciaer_nvr_rw_byte` structure pointed to by their arguments.~~

```
struct pciaer_nvr_rw_byte {  
    unsigned short address;  
    unsigned char data;  
};
```

~~In the read case, only the address must be valid on entry, and data will be filled in with the value read. The read may block (or return `-EAGAIN` in the non-blocking case) while the byte is read if it is not already cached. The write will block if and only if the `O_SYNC` flag is set. Attempting to read or write beyond the end of the NVRAM will return `-EINVAL`. The `...READ...` ioctl requires read access to the device; the `...WRITE...` ioctl requires write access.~~

~~Instead of using these two ioctls, applications should seek to the desired offset in the NVRAM and read or write a byte using the standard `read` and `write` calls.~~

PCIAER_IOC_GET_NVR_CACHE_STATUS

Fills the `int` pointed to by its argument with status flag bits. Initially only `PCIAER_IOC_NVR_CACHE_DIRTY` is defined. This ioctl requires read access to the device.

PCIAER_IOC_REVERT_NVR_CACHE

Resets the cache, marking all bytes which were previously marked dirty as not cached, hence 'undoing' all pending writes. No argument is used. This ioctl requires write access to the device.

Common ioctls

PCIAER_IOC_GET_VERSION_INFO

Fills the `pciaer_version_info` structure pointed to by its argument with the version number of the driver, the contents of the release registers of the two FPGAs and the contents of the S5920's revision identification register (RID). The FPGA release and driver version information is divided within the respective unsigned short into a high byte containing a major version number and low byte containing a minor revision number. This ioctl can be called on any of the minor devices and requires read access to the device.

```
struct pciaer_version_info {
    unsigned short driver_version;
    unsigned short fpga1_release;
    unsigned short fpga2_release;
    unsigned char s5920_revision_id;
};
```

PCIAER_IOC_SET_CNTR_PERIOD **PCIAER_IOC_GET_CNTR_PERIOD**

These two ioctls deal with the AER Clock Period. The `...SET...` ioctl uses its argument directly to specify the period in microseconds between counter updates. The only valid values are 1, 10, 50 and 100. The `...GET...` ioctl fills the `int` pointed to by its argument with one of these values according to the current status. These ioctls can be called on either Monitor or Sequencer. The `...SET...` ioctl requires write access to the device; the `...GET...` ioctl requires read access.

PCIAER_IOC_RST_CNTR_GET_TIME **PCIAER_IOC_GET_CNTR_LAST_RST_TIME**

These two ioctls deal with resetting the counter to 0 and can be called on either Monitor or Sequencer. The first (`...RST_CNTR_GET_TIME`) resets the counter and fills the `timeval` structure pointed to by its argument, if this is not NULL, with the time at which it did so. This ioctl requires read and write access to the device. The second ioctl (`...GET_CNTR_LAST_RST_TIME`) simply fills the `timeval` structure pointed to by its argument with the time at which the counter was last reset (without causing a reset). This requires read access to the device. The `timeval` structure is described on the *man* page for the `gettimeofday` system call. The last ioctl (`...RST_CNTR`) takes no argument and just resets the counter. This requires write access to the device.

PCIAER_IOC_SET_ARB_CONFIG **PCIAER_IOC_GET_ARB_CONFIG**

These two ioctls deal with how many devices are multiplexed into the arbiter. The `...SET...` ioctl uses its argument directly to specify one of the values `PCIAER_IOC_ARB_0_16`, `PCIAER_IOC_ARB_1_15` or `PCIAER_IOC_ARB_2_14`. The names of these constants reflect the way the 16 available bits are split between channel number and actual AE bits. The `...GET...` ioctl fills the `int` pointed to by its argument with one of these values according to the current status. These ioctls can be called on either Monitor or Mapper. The `...SET...` ioctl requires write access to the device; the `...GET...` ioctl requires read access.

PCIAER_IOC_SET_SEQ_ARB_CHANNEL **PCIAER_IOC_GET_SEQ_ARB_CHANNEL**

These two ioctls deal with connecting the sequencer output to one of the arbiter input channels. The `...SET...` ioctl uses its argument directly: an argument of 0, 1, 2 or 3 means connect the sequencer to the given arbiter input. The `...GET...` ioctl fills the `int` pointed to by its argument with 0, 1, 2 or 3 according to the current status. These ioctls can be called on either Sequencer or Mapper. The `...SET...` ioctl requires write access to the device; the `...GET...` ioctl requires read access.

PCIAER_IOC_RST_FIFO

This ioctl resets the FIFO of whichever device it is called on (Monitor, Sequencer, or Mapper), clearing any events that may be queued. No argument is used. This ioctl requires read access on the Monitor, or write access on the Sequencer or Mapper.

PCIAER_IOC_GET_PCI_INFO *(deprecated)*

This deprecated ioctl fills in the `pciaer_pci_info` structure pointed to by its argument. This enables the user to determine which PCI bus & slot any particular opened handle refers to, amongst other things. Note that if `slot_name` and `device_name` are non-NULL on entry, they must point to buffers to receive these strings, but it is not an error if either or both are NULL on entry – the corresponding string will simply not be available on exit. Furthermore, the string pointed to by `device_name` may not always be valid on exit, depending on the configuration of the kernel. If this is the case, the string will contain only a terminating nul character ('\0'). This ioctl can be called on any of the minor devices and requires read access to the device.

```
struct pciaer_pci_info {
    unsigned int bus;
    char *slot_name;
    char *device_name;
    unsigned short vendor;
    unsigned short device;
    unsigned short subsys_vendor;
    unsigned short subsys_device;
    unsigned char base_class;
    unsigned char sub_class;
    unsigned char prog_if;
    unsigned char irq;
    unsigned char slot;
    unsigned char func;
};
```

Instead of using this ioctl, applications can obtain all of this information (except for the device name) from the sysfs file system starting at `/sys/bus/pci/drivers/pciaer`.

PCIAER_IOC_GET_STATS *(deprecated)*

PCIAER_IOC_RST_STATS *(deprecated)*

These two [deprecated](#) ioctls deal with obtaining statistics from the Monitor, Sequencer or Mapper. Both take a pointer to a `pciaer_stats` [or](#) `pciaer_stats_2` structure as their argument (or optionally a NULL pointer in the case of the ...RST... ioctl). The first member of the structure must be set by the caller to indicate the size of the structure. If this value is not set correctly, -EINVAL will be returned. The ...GET... ioctl simply fills in the remaining elements of the structure. This ioctl requires read access to the device. The ...RST... ioctl resets the statistics relating to the minor device on which it is called. If its pointer argument is non-NULL, it also fills the structure with the statistics as they were prior to being reset. This ioctl requires write access to the device and also read access if the argument is non-NULL. The `pciaer_stats` [and](#) `pciaer_stats_2` structures [isare](#) defined as follows:

```
struct pciaer_stats {
    size_t size;
    unsigned long words_transferred;
    unsigned long user_transfers;
    unsigned long total_interrupts;
    unsigned long overflows_underflows;
    unsigned long timeouts;
    unsigned long memory_usage;
};
```

```
struct pciaer_stats_2 {
    size_t size;
    unsigned long words_transferred;
    unsigned long user_transfers;
    unsigned long total_interrupts;
    unsigned long overflows_underflows;
    unsigned long timeouts;
    unsigned long memory_usage;
    struct timeval timestamp;
};
```

The fields have the following meanings on the various minor devices:

	Monitor	Sequencer	Mapper
size	sizeof(struct pciaer_stats[2])	sizeof(struct pciaer_stats[2])	sizeof(struct pciaer_stats[2])
words_transferred	Number of words read by user process	Number of words written by user process	Reserved
user_transfers	Number of reads completed by user process	Number of writes completed by user process	Reserved
total_interrupts	Number of Monitor FIFO Half Full interrupts + number of Monitor FIFO Full interrupts	Number of Sequencer FIFO Half Empty interrupts + number of Sequencer FIFO Empty interrupts	Number of Mapper FIFO Full interrupts
overflows_underflows	Number of Monitor FIFO Full interrupts	Number of Sequencer FIFO Empty interrupts	Number of Mapper FIFO Full interrupts
timeouts	Number of internal timeouts	Reserved	Reserved
memory_usage	Internal buffer size in bytes	Internal buffer size in bytes	Reserved
<u>timestamp</u>	<u>Time at which statistics were read</u>	<u>Time at which statistics were read</u>	<u>Time at which statistics were read</u>

Instead of using the ioctls PCIAER_IOC_GET_STATS and PCIAER_IOC_RST_STATS, applications should obtain the same information via debugfs, see below.

Debugfs

If debugfs is available, the following files will appear there:

```
<debugfs>/pciaer/pciaer<n>/s5920/omb
<debugfs>/pciaer/pciaer<n>/s5920/imb
<debugfs>/pciaer/pciaer<n>/s5920/mbef
<debugfs>/pciaer/pciaer<n>/s5920/intcsr
<debugfs>/pciaer/pciaer<n>/s5920/rcr
<debugfs>/pciaer/pciaer<n>/s5920/ptcr
<debugfs>/pciaer/pciaer<n>/xilinx/rr1
<debugfs>/pciaer/pciaer<n>/xilinx/sra1
<debugfs>/pciaer/pciaer<n>/xilinx/cra1
<debugfs>/pciaer/pciaer<n>/xilinx/crb1
<debugfs>/pciaer/pciaer<n>/xilinx/crc1
<debugfs>/pciaer/pciaer<n>/xilinx/rr2
<debugfs>/pciaer/pciaer<n>/xilinx/sra2
<debugfs>/pciaer/pciaer<n>/xilinx/cra2
<debugfs>/pciaer/pciaer<n>/mapstats
<debugfs>/pciaer/pciaer<n>/monstats
<debugfs>/pciaer/pciaer<n>/seqstats
```

where **<debugfs>** is the path to the debugfs mount point (usually /sys/kernel/debug) and **<n>** is the pciaer board number.-

Each of [the first 14 of](#) these files allows the corresponding S5920 [2] or Xilinx FPGA [1] based register to be read, and if appropriate to be written to, for debugging purposes. The register values read from these files are formatted according to %08X; values written will be assumed to be in hexadecimal.†

[The last three files can be read to obtain statistics relating to the mapper, monitor and sequencer respectively. For each device, the statistics are formatted as a series of lines each consisting of a decimal number followed by a space and one of the struct pciaer_stats_2 field names. See the table in the preceding section for a description of the meanings of these fields. Writing to one of these files will reset the corresponding statistics.](#)

References

1. PCI-AER Adapter board User Manual (Rel 1.1), [Vittorio Dante, Istituto Superiore di Sanità, Rome, Italy](#), 5 Nov 2004 .
2. AMCC PCI Products Data Book, SECTION 2: S5920 PCI Target Interface, Applied Micro Circuits Corporation, 6290 Sequence Drive, San Diego, CA 92121-4358, <http://www.amcc.com>.
3. [Corbet, Jonathon, Rubini, Alessandro, and Kroah-Hartman, Greg, "Linux Device Drivers", 3rd edn. ISBN 0-596-00590-3, O'Reilly, 2005.](#)
4. ["LINUX ALLOCATED DEVICES", maintained by Torben Mathiasen, Documentation/devices.txt within the Linux kernel source tree or http://www.lanana.org/docs/device-list/.](#)
5. [Man tty_ioctl](#)

Appendix – ioctl Table

The following table lists all defined ioctls in ordinal number order.

Symbol	Macro	Ord.	Data item	Req'd perm.	Comments
PCIAER_IOC_GET_NVRAM_SIZE		48			Obsolete – the same information can be obtained using <code>lseek</code> .
PCIAER_IOC_READ_NVR_BYTE	__IOWR	49	struct pciaer_nvr_rw_byte	r--	Deprecated <i>Obsolete</i> – use read instead
PCIAER_IOC_WRITE_NVR_BYTE	__IOW	50	struct pciaer_nvr_rw_byte	-w-	Deprecated <i>Obsolete</i> – use write instead
PCIAER_IOC_GET_NVR_CACHE_STATUS	_IOR	51	int	r--	
PCIAER_IOC_REVERT_NVR_CACHE	_IO	52		-w-	
PCIAER_IOC_GET_PCI_INFO	_IOR	56	struct pciaer_pci_info	r--	Deprecated – use <code>sysfs</code> instead.
PCIAER_IOC_RST_XILINX	_IOW	57	int	-w-	Reserved but not implemented
PCIAER_IOC_SET_SEQ_ARB_CHANNEL	_IO	65		-w-	
PCIAER_IOC_GET_SEQ_ARB_CHANNEL	_IOR	66	int	r--	
PCIAER_IOC_SET_MON_CH_SEL	_IO	67		-w-	
PCIAER_IOC_GET_MON_CH_SEL	_IOR	68	int	r--	
PCIAER_IOC_SET_MAP_MAPPING	_IOW	69	struct pciaer_mapping	-w-	
PCIAER_IOC_GET_MAP_MAPPING_COUNT	_IOWR	70	struct pciaer_mapping	r--	
PCIAER_IOC_GET_MAP_MAPPING	_IOWR	71	struct pciaer_mapping	r--	
PCIAER_IOC_MAP_ADD_TO_MAPPING	_IOW	72	struct pciaer_mapping	-w-	Previously PCIAER_IOC_ADD_MAP_MAPPING
PCIAER_IOC_MAP_DEL_FROM_MAPPING	_IOW	73	struct pciaer_mapping	-w-	Previously PCIAER_IOC_DEL_MAP_MAPPING
PCIAER_IOC_FIND_NEXT_MAP_MAPPING	_IOWR	74	int	r--	
PCIAER_IOC_GET_MAP_MAPPED_MAP	_IO	75		r--*	
PCIAER_IOC_SET_MON_TIME_LBL_FLAG	_IO	76		-w-	
PCIAER_IOC_GET_MON_TIME_LBL_FLAG	_IOR	77	int	r--	
<i>PCIAER_IOC_GET_MAP_FIFO_FILLS</i>		78			<i>Obsolete. Use PCIAER_IOC_GET_STATS</i>
PCIAER_IOC_GET_STATS	_IOWR	78	struct pciaer_stats	r--	<i>Deprecated</i> – use <i>debugfs</i> instead.
<i>PCIAER_IOC_RST_MAP_FIFO_FILLS</i>		79			<i>Obsolete. Use PCIAER_IOC_RST_STATS</i>
PCIAER_IOC_RST_STATS	_IOWR	79	struct pciaer_stats	?w-*	<i>Deprecated</i> – use <i>debugfs</i> instead. Read permission req'd iff arg != NULL
PCIAER_IOC_SET_MAP_OUT_CONFIG	_IO	80		-w-	
PCIAER_IOC_GET_MAP_OUT_CONFIG	_IOR	81	int	r--	
<i>PCIAER_IOC_RST_CNTR</i>		82			<i>Obsolete. Use PCIAER_IOC_RST_CNTR_GET_TIME with arg = NULL instead.</i>
PCIAER_IOC_RST_CNTR_GET_TIME	_IOR	82	struct timeval	?w-*	Read permission req'd iff arg != NULL
PCIAER_IOC_SET_MAP_DEMUX_CONFIG	_IO	83		-w-	
PCIAER_IOC_GET_MAP_DEMUX_CONFIG	_IOR	84	int	r--	
PCIAER_IOC_SET_MAP_AER_PROTOCOL	_IO	85		-w-	
PCIAER_IOC_GET_VERSION_INFO	_IOR	86	struct	r--	

Symbol	Macro	Ord.	Data item	Req'd perm.	Comments
			pciaer_version_info		
PCIAER_IOC_GET_MAP_AER_PROTOCOL	_IOR	87	int	r--	
PCIAER_IOC_MAP_CLEAR	_IO	88		-w-	
PCIAER_IOC_SET_MAP_CH_SEL	_IO	89		-w-	
PCIAER_IOC_MAP_COMPACT	_IO	90		-w-	
PCIAER_IOC_GET_MAP_FREE_SPACE	_IOR	91	unsigned int	r--	
PCIAER_IOC_GET_CNTR_LAST_RST_TIME	_IOR	92	struct timeval	r--	
PCIAER_IOC_GET_MON_DATA_WORD		97			Obsolete – use read instead.
PCIAER_IOC_ENABLE_MAP_OUT	_IO	98		-w-	
PCIAER_IOC_DISABLE_MAP_OUT	_IO	99		-w-	
PCIAER_IOC_GET_MAP_OUT_STATE	_IOR	100	int	r--	
PCIAER_IOC_GET_MAP_FIFO_FLAGS	_IOR	101	int	r--	
PCIAER_IOC_GET_MON_FIFO_FLAGS	_IOR	102	int	r--	
PCIAER_IOC_GET_SEQ_FIFO_FLAGS	_IOR	103	int	r--	
PCIAER_IOC_RST_FIFO	_IO	104		??-*	Required permission: mon r--; seq -w-; map -w-
PCIAER_IOC_SET_SEQ_IN_ARB		105			Superseded by PCIAER_IOC_SET_SEQ_ARB_CHANNELNEL
PCIAER_IOC_GET_SEQ_IN_ARB		106			Superseded by PCIAER_IOC_GET_SEQ_ARB_CHANNELNEL
PCIAER_IOC_SET_CNTR_PERIOD	_IO	107		-w-	
PCIAER_IOC_GET_CNTR_PERIOD	_IOR	108	int	r--	
PCIAER_IOC_SET_ARB_CONFIG	_IO	109		-w-	
PCIAER_IOC_GET_ARB_CONFIG	_IOR	110	int	r--	
PCIAER_IOC_DEBUG_XILINX_READ		111			Obsolete – use debugfs instead.
PCIAER_IOC_DEBUG_XILINX_WRITE		112			Obsolete – use debugfs instead.
PCIAER_IOC_DEBUG_S5920_READ		113			Obsolete – use debugfs instead.
PCIAER_IOC_GET_MON_DATA_BLOCK		114			Obsolete – use read instead. Ordinal previously also used for deprecated CFG_IOC_FREIDL
PCIAER_IOC_GET_MON_FIFO_DEPTH		115			Obsolete – use PCIAER_IOC_GET_FIFO_DEPTH. Ordinal previously also used for deprecated CFG_IOC_SWRITE
PCIAER_IOC_GET_FIFO_DEPTH		116			Obsolete - use fstat instead - see text for details.
PCIAER_IOC_DEBUG_FIFO_WRITE		117			Obsolete – use Sequencer write instead.
PCIAER_IOC_DEBUG_SRAM_READ		118			Obsolete – use Mapper mmap instead.
PCIAER_IOC_DEBUG_S5920_WRITE		119			Obsolete – use debugfs instead.
PCIAER_IOC_DEBUG_FIFO_READ		120			Obsolete – use Monitor read instead.
PCIAER_IOC_GET_MAP_CH_SEL	_IOR	121	int	r--	
PCIAER_IOC_DEBUG_SRAM_WRITE		122			Obsolete – use Mapper mmap and mprotect instead.

A '*' in the Req'd perm. column indicates an ioctl which does not follow the regular pattern of ioctls defined using _IOR or _IOWR requiring read permission and those defined using _IO or _IOW requiring write permission.