ATLAS Internal Note
ATL-X-XX-NNNN
Version 1.5
30 January 2018

The “flx” device driver

Authors : M. Joos, J. Vermeulen

comments and queries to Markus Joos, CERN
+41 22 767 2364
Markus.Joos@cern.ch

Abstract

This note describes the user interface of the “flx” device driver that has been developed in order to control the FELIX PCIe I/O cards that will be used in the ATLAS Read-Out system.

Introduction

The “flx” driver is a conventional device driver for a PCIe device. It supports the detection of FELIX cards on the basis of the DID/VID, handles interrupts and provides a number of ioctl functions. One of these functions (the SETCARD function) allows user processes to obtain physical addresses for the resources of the FELIX card. It is therefore possible to develop user S/W that manages the FELIX cards without context switching for simple I/O operations.

The driver is work in progress and will have to track the evolution of the FELIX H/W. However, the dependency on firmware / hardware properties is small.

It is not yet known if the driver in its current form is protected from race conditions that may arise from concurrent accesses from multiple, independent processes.

In the ATLAS framework, the flx driver will usually be used together with the cmem_rcc driver and (optionally) the io_rcc driver. Documentation for these drivers is available from M. Joos.

Capabilities of the file /proc/flx

The “flx” driver creates a file with the name “`flx`” in the `/proc` file system. When read (e.g. “`more /proc/flx`”) the file provides information about the status of the hardware. Writing to the file (e.g. “`echo debug >/proc/flx`”) enables the user to change e.g. the level of verbosity of the debug output. See below for details.

Parameters of the driver

The flx driver can be configured by the user within certain limits. The table below specifies the supported parameters and their meaning as well as the user interface to them.

Name of the parameter	**User interface**	**Description**
debug	/proc/flx and insmod	Setting the parameter to “1” enables verbose debug output to /var/log/messages. This can be done as follows: • insmod flx.ko debug=1 • echo debug >/proc/flx Setting the parameter to “0” disables verbose debug output to /var/log/messages. This can be done as follows: • insmod flx.ko debug=0 • echo nodebug >/proc/flx Default: disabled
errorlog	/proc/flx and insmod	Setting the parameter to “1” enables verbose error output to /var/log/messages. This can be done as follows: • insmod flx.ko errorlog=1 • echo elog >/proc/flx Setting the parameter to “0” disables verbose error output to /var/log/messages. This can be done as follows: • insmod flx.ko errorlog =0 • echo noelog >/proc/flx Default: enabled

msiblock	insmod	This parameter controls the number of MSI-X interrupts that the FELIX card can handle. It can be selected in the range 1 – 8. Default: 8

Interrupt capabilities

The FELIX cards use MSI-X interrupts in order to signal asynchronous events to the operating system. Per installed card, the driver supports up to 8 interrupts, enumerated from 0 to 7. The number of interrupts is specified in the source of the driver by a constant (MAXMSI) and is determined by the firmware. There is no specific dependence of the driver on the nature of the interrupts. When the driver gets loaded into the kernel (this usually happens at boot time) all interrupts are by default disabled. The user library (FlxCard API) provides a method for enabling interrupts.

Description of the `ioctl` functions

Developers of user code are advised not to directly interface to the driver (i.e. not to call ioctl functions) but to use the Low-Level API. For details, refer to this document.

Internal implementation (not relevant for users)

The `ioctl` function is called with a file descriptor, the `ioctl` name and, if applicable, a user parameter. After opening the driver, the first `ioctl` function to be called is the `SETCARD` function, with a pointer to a `struct` of type `card_params_t` as user parameter. The card number has to be stored in this `struct` before calling the `ioctl SETCARD` function. The function copies the BAR0, BAR1 and BAR2 addresses as well as the sizes of the memory areas associated with these base addresses to the `struct`. The addresses are physical addresses. After mapping to virtual addresses the registers can be addressed directly from used space.

The file descriptor also contains a pointer ("`private_data`") to a second `struct` of type `card_params_t`. The same information as stored in the `struct` mentioned above is also stored in this `struct` by the `ioctl SETCARD` function, so that the file descriptor is associated with a card number and also with the information on the memory areas (NB: the information on the memory areas is not used in the driver, so the pointer could as well point to a single integer with the card number). The card number is set to 0 when opening the driver.

An interrupt causes a flag associated with it to be set and a process can wait for this to occur. By setting all flags with the `ioctl CANCEL_IRQ_WAIT` function, any process waiting for an interrupt will be woken up.

The IOCLT entry points

CANCEL_IRQ_WAIT--

`ioctl` name	`CANCEL_IRQ_WAIT`

Type of user parameter	int *
Purpose of user parameter	Send data to the driver

Description:

This function wakes up processes currently waiting for the interrupt with the number specified in the integer pointed to from the FELIX card by setting the corresponding interrupt flag.

CLEAR_IRQ--

`ioctl` name	`GET_IRQ_COUNTER`
Type of user parameter	int *
Purpose of user parameter	Send data to the driver

Description:

This function clear unsolicited interrupts. If the H/W has sent interrupts while the S/W is not yet ready to handle them, the interrupts will “pile up” in the driver. A call to CLEAR_IRQ clear old inetrrupts.

GETCARDS--

`ioctl` name	`GETCARDS`
Type of user parameter	`int*`
Purpose of user parameter	Receive data from the driver

Description:

This function returns the number of FELIX cards that have been found in the computer in the `int` variable pointed to by the user parameter.

GET_TLP--

`ioctl` name	`GET_TLP`
Type of user parameter	`int*`
Purpose of user parameter	Receive data from the driver

Description:

Provides the maximum TLP size (for background information see for example "Down to the TLP: How PCI express devices talk", http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1) in multiples of 128 bits as obtained from the configuration space. The TLP size is needed for initialization of the DMA controller. Recently (15 May 2017) the user API was changed and the TLP size is no longer a parameter. Therefore the GET_TLP ioctl will be called only from the FlxCard library.

MASK_IRQ--

`ioctl` name	`MASK_IRQ`
Type of user parameter	`int*`
Purpose of user parameter	Send data to the driver

Description:

This function disables the interrupt with the number specified in the integer variable pointed to by the user parameter.

UNMASK_IRQ--

`ioctl` name		`UNMASK_IRQ`
Type of user parameter		`int*`
Purpose of user parameter		Send data to the driver

Description:

This function enables the interrupt again after a call of the `ioctl MASK_IRQ` function with the number specified in the `int` variable pointed to by the user parameter.

RESET_IRQ_COUNTERS--

`ioctl` name	`RESET_IRQ_COUNTERS`
Type of user parameter	`int*`
Purpose of user parameter	Send data to the driver

Description:

With each interrupt a counter is associated that counts the number of interrupts observed. The count is displayed in `/proc/flx`. This function resets the counter with the number specified in the integer variable pointed to by the user parameter.

WAIT_IRQ--

`ioctl` name	`WAIT_IRQ`

Type of user parameter	`int*`
Purpose of user parameter	Send data to the driver

Description:

The process calling this function will be put to sleep until the flag associated with the interrupt with the number specified in the `int` variable pointed to by the user parameter is set. The function resets the flag after the process was woken up.

SETCARD--

`ioctl` name	`SETCARD`
Type of user parameter	pointer to `struct` of type `card_params_t`
Purpose of user parameter	Send data to and receive data from the driver

Description:

The user specifies the card number (enumerated from 0 to N-1) of a FELIX card in the "`slot`" variable of the `card_params_t` structure. Then it calls the `ioctl` function. The driver fills the other variables (e.g. the PCI bas addresses) of the `card_params_t` structure with data, returns the structure to the user and associates the card number with the file descriptor, see also the beginning of this section.

As from version 3.0.0 (RPM version) of the driver the user can also set the variable `lock_mask` in `card_params_t`. It defines which resources of the FLX card (e.g. DMA channels or I2C interfaces) must only be accessed by the calling process. See the API document for further information.

GETLOCK--

`ioctl` name	`GETLOCK`
Type of user parameter	pointer to `u_int`
Purpose of user parameter	Send data to and receive data from the driver

Description:

This ioctl can be used to retrieve the locking bits for a given FLX card.

RELEASELOCK--

`ioctl` name	`RELEASELOCK`
Type of user parameter	pointer to `struct` of type `lock_params_t`
Purpose of user parameter	Send data to the driver

Description:

This ioctl allows an application to release locked resources.

To be done:

- discuss which commands should be implemented via the proc_write interface (such as enabling debugging or clearing interrupts)
- Add text about the “global interrupt strategy”
    - What types of interrupts exist?
    - What are typical use cases?
    - How are stacked interrupts dealt with?
    - Can two processes wait for the same interrupt?
    - How do interrupts get acknowledged? Is it done automatically by the driver or can there be cases where the acknowledgement has to be done in the user code?