The FILAR driver and library

Author: Markus Joos, CERN
Date: 2. Dec. 2002
Version: 1.1

1. Introduction

This document describes the Linux driver and library developed for the FILAR F/W on the future FILAR H/W but it can also be used on a S32PCI64 H/W with FILAR F/W. In the following this latter combination of H/W and firmware will be called the "S32PCI64-FILAR". This term will be used for features specific to the FILAR F/W on the S32PCI64 H/W. The term "FILAR" will be used if the respective feature applies to both today's S32PCI64-FILAR and the future FILAR.

2. The S32PCI64 H/W

The S32PCI64 is a 64 bit / 66 MHz PCI card with one CMC site for 3.3V S-Link LDC cards. It can be purchased from Nowoczesna Elektronika [1]. For further information see [2].

3. The FILAR F/W

The FILAR is a future S-Link product which will be derived from the S32PCI64. In contrast to the S32PCI64 it will have 4 on-board HOLA based S-Link receivers. Its PCI interface will be the one of the S32PCI64. The programming model of the FILAR will be an improved version of the protocol originally designed for the S32PCI64. The main difference is the reduction of PCI single cycles (hand shaking protocol) per S-Link packet from five to two. In order to test the FILAR concept and to improve the performance of the existing S32PCI64 hardware it has been decided to develop a FILAR emulation F/W that turns the S32PIC64 into something like a single channel FILAR. Even though originally meant to be an intermediate step in the development of the final FILAR this F/W turned out to be the better choice with respect to the F/W originally developed for the S32PCI64 due to its lower overhead. Users of the S32PCI64 H/W are therefore recommended to install the FILAR (emulation) F/W on their cards.

4. The software

4.1. Design principles

The design of the driver and library was based on the considerations listed below:

- The code handshaking the FILAR may have to co-exist on one CPU with the other processes. In such an environment it would be unlikely that the S/W could react quickly enough to requests from the H/W if it was a polling based user process relying on the Linux scheduler. The better choice is to use interrupts and have them handled by a dedicated driver.
- A "classical" user library based on polling would add an additional load to the PCI. This is an other reason for basing the driver on interrupts.
- The deep REQ and ACK FIFOs of the FILAR allow to process events in bursts. This reduces the interrupt frequency a lot.
- In order to minimize the amount of data copied between user space and the driver a shared memory region (accessible to both the user application and the driver) has been implemented for the exchange of data for the S/W FIFOs (see below).

4.2. FIFOs

The control of the FILAR is based on two pairs of FIFOs. The first pair are the H/W FIFOs (REQ and ACK) of the S32PCI64-FILAR. These FIFOs allow the FILAR to receive up to 32 S-Link packets without any S/W intervention.

The second pair are S/W FIFOs (IN and OUT) managed by the driver. One can view them as an extension of the H/W FIFOs. The depth of these FIFOs is a parameter that can be set by the user (in a header file). As long as these FIFOs are not empty / full the FILAR will only need services from the interrupt service routine in the driver to continue inputting S-Link packets. Only once these FIFOs have reached their limits does the user process have to process the new packets and provide resources for further data.

4.3. The driver

The control of the FILAR is performed by a driver. It provides a number of ioctl()entry points for communication with the library and a (large) interrupt service routine (ISR) for the handshaking of the FILAR registers. This ISR is triggered by an interrupt sent by the FILAR if its ACK FIFO of one active channel has at least 24 entries. Once started the ISR handshakes all S32PCI64-FILAR cards installed. This is to reduce the interrupt frequency at the price of an extended latency of the ISR.

During its initialization phase the driver scans the entire PCI space for FILAR cards and puts them into an internal list. As it has been developed for the final FILAR it assumes four channels (0..3) per PCI card. In the case of a S32PCI64-FILAR only channel 0 must be used.

Once installed the driver provides the file /proc/filar which contains information about the number of S32PCI64-FILAR cards installed and their status.

4.4. The library

This chapter describes the user interface to the FILAR. As it has been designed for the final FILAR cards (with four channels and some additional features) it contains some functions / parameters which are not available with the S32PCI64-FILAR. The library provides the functions listed below.

Synopsis

FILAR_ErrorCode_t FILAR_Open(void)

Parameters

None

Description

This function initializes the package. It can be called several times. Only the first call performs the open action. Subsequent calls increment a counter.

Errors

FILAR_ERROR_FAIL	The rcc_error library did not open
FILAR_FILE	The library did not manage to open the file /dev/filar
FILAR_EFAULT	The driver failed in the copy_to_user() call
FIALR_MMAP	The driver failed to map some shared memory to user space

---

Synopsis

FILAR_ErrorCode_t FILAR_Close(void)

Parameters

none

Description

This function closes the library. It has to be called as many times as FILAR_Open. Only the last call closes the library.

Errors

FILAR_NOTOPEN	The library has not been opened yet
FILAR_MUNMAP	The driver failed to un-map some shared memory from user space

---

Synopsis

FILAR_ErrorCode_t FILAR_Init(FILAR_contig_t *config)

Parameters

FILAR_contig_t *config	in	A structure with initialization parameters for the FILAR modules

Description

This function collectively initializes all the FILAR modules installed in the PC. The structure "FILAR_config_t" (see filar_common.h) is defined as follows:

```
{
u_int enable[MAXROLS];     // To enable / disable channels
u_int psize;               // The value for bits 5-3 of the OPCTL register.
                           // See [ 3]
u_int bswap;               // Byte swapping. Not implemented for the
                           // S32PCI64-FILAR
u_int wswap;               // Word swapping. Not implemented for the
                           // S32PCI64-FILAR
u_int interrupt;           // The value for bit 1 of the IMASK register. Has to
                           // be set to 1. See [ 3]
u_int scw;                 // Start control word. For checking. Not implemented
                           // for the S32PCI64-FILAR
u_int ecw;                 // End control word. For checking. Not implemented
                           // for the S32PCI64-FILAR
}
```

The parameter *enable* is an array of *MAXROLS* integers. Each element defines if the respective channel is enabled (=1) or disabled (=0). If multiple FILAR cards are installed the array index (= channel number) has to be computed with the formula:

channel = (# of FILAR card) * 4 + (number of channel)

Both numberings start at 0.
Example: The third channel on the second FILAR card can be enabled with *enable[6]*.
In case of S32PCI64-FILAR cards the only channels 0, 4, 8, etc. are available.

The parameter *psize* defines the page size for all FILAR cards installed. See [3].

Errors

FILAR_NOTOPEN	The library has not been opened yet
FILAR_NOHW	One of the channels which were set to 1 in *enable[]* does not exist

---

Synopsis

FILAR_ErrorCode_t FILAR_Info(FILAR_info_t *info)

Parameters

FILAR_intfo_t *info	in	A structure with information about the installed H/W

Description

This function returns information about the hardware (i.e. FILAR cards) installed in the PC. The structure *FILAR_info_t* is defined as follows:

```
{
u_int nchannels;
u_int channels[MAXROLS];
u_int enabled[MAXROLS];
}
```

The parameter *nchannels* reports the total number of channels available on the FILAR cards in the PC. In case of a S32PCI64-FILAR it reports 4 channels per PCI card. Only channel 0 can be used for S-Link I/0. Channels 1..3 are emulated and repeat the data received on channel 0.

The array *channels* reports if a given channel is present (=1) or absent (=0).

The array *enabled* reports if a given channel is enabled (=1) or disabled (=0).

The constant MAXROLS can be modified in a header file (filar_common.h). It defines how many channels the driver / library can handle.

Errors

FILAR_NOTOPEN	The library has not been opened yet
FILAR_EFAULT	The driver could not execute the copy_to_user call

---

Synopsis

FILAR_ErrorCode_t FILAR_PagesIn(FILAR_in_t *fifoin)

Parameters

FILAR_in_t *fifoin	in	A structure with PCI addresses for the REQ FIFO

Description

This function fills the IN FIFO of the driver with PCI addresses to which the data of future S-Link packets will be written. The driver automatically reads these addresses from the IN FIFO and writes them to the REQ FIFO of the FILAR H/W as required. The structure FILAR_in_t is defined as follows:

```
{
u_int nvalid;
u_int channel;
u_int pciaddr[MAXINFIFO];
}
```

The parameter *nvalid* has to be equal to the number of valid PCI addresses in the *pciaddr* array.

The parameter *channel* defines for which channel the PCI addresses are meant. The formula to compute channel from the number of the PCI card and the number of the channel on that card is:

channel = (# of FILAR card) * 4 + (number of channel)

Both numberings start at 0.

The array *pciaddr* has to be filled with up to MAXINFIFO PCI addresses. The FILAR will write incoming S-Link packets to these addresses in the order in which they appear in the array. These PCI addresses have to point to contiguous memory blocks of a size equal to the parameter *psize* in the function FILAR_Init.

Errors

FILAR_NOTOPEN	The library has not been opened yet
FILAR_NOHW	The channel selected with the parameter *channel* does not exist

FILAR_EFAULT	The driver could not execute the copy_from_user call
FILAR_FIFOFULL	The IN FIFO of the respective channel became full. Not all PCI addresses have been written.

Synopsis

FILAR_ErrorCode_t FILAR_PagesOut(u_int channel, FILAR_out_t *fifoout)

Parameters

u_int channel	in	The number of the channel to be interrogated
FILAR_out_t *fifoout	out	A structure with information about recently received S-Link packets

Description

This function reads back information about S-Link packets that have arrived since the last call to that function. The structure FILAR_out_t is defined as follows:

```
{
u_int nvalid;
u_int pciaddr[MAXOUTFIFO];
u_int fragsize[MAXOUTFIFO];
u_int fragstat[MAXOUTFIFO];
}
```

The parameter *channel* is an input value. It has to be set by the application to the number of the channel to be read out. See FILAR_PagesIn for how to compute the channel number.

The parameter *nvalid* indicates the number of S-Link packets that have arrived for the selected channel.

The parameters *pciaddr*, *fragsize* and *fragstat* return the PCI address, S-Link packet size (in 32-bit words) and fragment status for the respective S-Link packets. If a S-Link packet, as sent by the S-Link source, is larger than the page size (see parameter *psize* in FILAR_Init) the FILAR will break this packet up into several fragments and report them individually via the fifoout structure. Errors in a packet are reported via fragstat which is a packed integer consisting of boolean flags defined in filar_common.h.

NOTE:
This function is based on the principle of polling which is adequate for a system with a high activity on S-Link where it is likely that a call to FILAR_PagesOut returns some new packets. In case of a mostly idle link FILAR_PagesOut, if called at a high frequency, will mostly return *nvalid*=0 indicating that there are no new packets. The overhead for an idle call is small as this function does not have to call a driver service routine. In principle it would be possible to instruct the ISR of the driver not only to service the FILAR but also to send a signal to a user process. This feature can be added if required.

Errors

FILAR_NOTOPEN	The library has not been opened yet
FILAR_NOHW	The channel selected with the parameter *channel* does not exist

---

Synopsis

FILAR_ErrorCode_t FILAR_Reset(u_int channel)

Parameters

u_int channel	in	The number of the channel to reset

Description
This function is meant to reset a FILAR card. As the library does not have a notion of the card number the parameter *channel* is used to represent the card this channel resides on. The action performed is a card reset as specified for bit 0 of the OPCTL register. Additionally all S/W FIFOS for this card are reset.

Errors
FILAR_NOTOPEN	The library has not been opened yet
FILAR_NOHW	The channel selected with the parameter *channel* does not exist
FILAR_EFAULT	The driver could not execute the copy_from_user call

---

Synopsis
FILAR_ErrorCode_t FILAR_LinkReset(u_int channel)

Parameters
u_int channel	in	The number of the channel to reset

Description
This function is meant to reset a single S-Link channel. It sets the URESET bit of the respective channel, waits for the LDOWN of that channel to come up and resets the URESET. After this operation the link should be up.

Errors
FILAR_NOTOPEN	The library has not been opened yet
FILAR_NOHW	The channel selected with the parameter *channel* does not exist
FILAR_STUCK	The respective link did not come up again after the reset
FILAR_EFAULT	The driver could not execute the copy_from_user call

---

Synopsis
FILAR_ErrorCode_t FILAR_LinkStatus(u_int channel, u_int *status)

Parameters
u_int channel	in	The number of the channel to reset
u_int *status	out	“0” if the Link is down, “1” if the link is up

Description
This function checks the status of link of the respective channel. The return value *status* indicates if the link is down (=0) or up (=1).

Errors
FILAR_NOTOPEN	The library has not been opened yet
FILAR_NOHW	The channel selected with the parameter *channel* does not exist
FILAR_EFAULT	The driver could not execute the copy_from_user call

---

Synopsis
FILAR_ErrorCode_t FILAR_InFree(u_int channel, u_int *nfree)

Parameters
u_int channel	in	The number of the channel to be checked
u_int *nfree	out	The number of free slots in the IN FIFO of the selected channel

Description
This function can be used to determine the number of free slots in the IN FIFO of a channel. This number is required to prevent sending to many PCI addresses to the FILAR via FIFO_PagesIn.

Errors
FILAR_NOTOPEN	The library has not been opened yet
FILAR_NOHW	The channel selected with the parameter *channel* does not exist

Synopsis
FILAR_ErrorCode_t FILAR_Flush(void)

Parameters
None

Description
This function triggers the execution of the ISR in the driver. It can be called at any time. Its purpose is to flush remaining events from the ACK FIFO of the FILAR at the end of a data talking session. See 5.1.

Errors
FILAR_NOTOPEN	The library has not been opened yet

5. Programming issues

5.1. Visibility of newly arrived packets

Even though the FILAR writes the data of new packets directly to the memory of the PC it is not possible for the S/W to "see" the data immediately. The "ACK-FIFO almost full" interrupt implicitly causes the system to always process packets in groups. This means that there is a dead-time during which newly arrived packets are invisible because the information about them has not been read out of the ACK FIFO. The length of this dead-time depens on the rate at which new packets arrive on S-Link.
The bigger problem is that at the end of a data taking session some packets may still be stuck in the ACK FIFO as due to the absence of new packets the FIFO will not pass the interrupt threshold and the driver will therefore never process the last packets. A solution to this problem is the function FILAR_Flush() that fakes trigger the execution of the ISR in the driver and thereby forces the driver to process the last packets.

5.2. Deadlocks

The PCI interrupt sent by the FILAR once an ACK FIFO has passed the threshold stays active as long as the "almost full" condition is true for the FIFO. The ACK FIFO, as described above, is read out by the ISR of the driver. Before reading the ACK FIFO this function checks if there is space in the OUT FIFO to which the information found in the ACK FIFO will have to be written. If the OUT FIFO is full the ISR can not read the ACK FIFO and hence it can not clear the interrupt. The system is therefore in a deadlocked state. The only safe way to get out of this is to disable the interrupt on the FILAR card. In such a case, the driver writes an error message to /var/log/messages. The status of the interrupt can also be found in /proc/filar.
In order to avoid this deadlock one must be sure that there is always space in the OUT FIFO. This is achieved by assuring that the application program does not send more PCI addresses to the FILAR (FILAR_PagesIn) than the OUT FIFO can handle.

5.3. How to configure a S32PCI64-FILAR

The S32PCI64-FILAR is special in the sense that only one of its four channels (0) can be used for real data transfers. The other three channels (1..3) only copy the data of the first channel. If used for data talking the channels 1..3 of the S32PCI64-FILAR have to be disabled (see FILAR_Init) before the first S-Link packet is received.

5.4. PCI card numbering

As mentioned above the library uses a flat scheme to number S-Link channels and a simple formula to compute the channel from the PCI card number. This number is not something that the user can control. The number of a PCI card is determined by the PCI scanning algorithm of Linux. If several identical (same vendor and device ID) cards are installed in one PC the card which is first found by the PCI scan is referred to as card 0, the card found in second place is card 1. In which order the cards are found depends on the device numbers of the slots they are installed in an on the hierarchy of PCIs on the PC motherboard. Please consult the documentation for your PC for further information.

5.5. Error handling

The functions in this package return error according to the format of the RCC_ERROR package. This package, in turn, is, apart from the name, identical to the IOM_ERROR package of the ATLAS DAQ-1 project. See [4] for documentation.

5.6. Link reset

So far the package only contains a function to reset an entire FILAR card. Technically it is also possible to reset a single S-Link channel. A function supporting this feature can be added on request.

5.7. Card temperature

The final FILAR will have an on-board sensor to measure the temperature of the card. As this feature does not exist in the S32PCI64 H/W the function supporting it is still missing.

6. Support

This package has been developed specifically for the ATLAS experiment at CERN. Some of the concepts and techniques used may not be adequate for other applications. The author is open for any type of feedback and will add features on a best effort basis in as far as they do not collide with the main purpose of this package (i.e. ATLAS). As the package is distributed in source form users may adapt the code to their requirements. The package has been tested on a RedHat 7.2 system under kernel version 2.4.9 with single S32PCI64-FILAR cards. Tests with multiple cards are yet to be done.

7. References

[1] **http://www.no-el.krakow.pl/products.html**

[2] **http://hsi.web.cern.ch/HSI/s-link/devices/s32pci64**

[3] **http://edms.cern.ch/document/337904/1**

[4] **http://atddoc.cern.ch/Atlas/Notes/051/Note051-1.html**