FMC ADC User's Manual

September 2014 (fmc-adc-100m-sw-2014-05-44-g91e47c2-dirty)
FMC ADC 100M 14b 4ch – software manual

CERN BE-CO-HT / Federico Vaga / Alessandro Rubini

Table of Contents

Introduction

This is the user manual of the driver for the *FMC ADC 100M 14b 4cha* board developed on the **Open Hardware Repository**[1]. *FMC* is the form factor of the card, *ADC* is its role, *100M* means it can acquire 100Msample per second, *14b* is the numbers of meaningful bits and *4cha* states it has 4 input channels (plus a trigger input).

If you want to start acquiring straight off, we suggest running *fald-simple-ack*, described in Chapter 13 [Simple Acquisition], page 15.

1 Bugs and Missing Features

To set the record straight, we'd better list the known issues right off at the start. Release 1.1 will follow shortly (summer 2013), dealing with most of these details. Meanwhile, the package (hardware, gateware, software) is rock solid, so we release it despite this list of known deficiencies:

- The driver doesn't check acquisition size overflows (the code is almost there but it is not triggering). Each shot in multi-shot is limited to 2048 samples, and a single-show acquisition is limited to 32M samples.
- Some attributes should probably be renamed, to use human-readable names and numbers, instead of exposing hardware-internal values. This applies to *vref/range* mainly.
- Software trigger (e.g. ZIO *timer* trigger) works properly, but the sequence number of blocks is incrementing by 2 at each block. This must be fixed in this driver, that increments the sequence number at the wrong place.
- Timestamps generated by the hardware are not configured nor properly used.
- Some error messages in the tools are puzzling and should be fixed.
- The library has a number of issues too, but the fix won't have user-visible effect (i.e. no incompatibilities in the API).
- API documentation is incomplete, especially about "parameters", internals and portability issues.

2 Repositories and Releases

This project is hosted on the Open Hardware repository at the following link:

`http://www.ohwr.org/projects/fmc-adc-100m14b4cha-sw`

Here a list of resources that you can find on the project page.

`Document`[2] contains the `.pdf` documentation for every official release.

`File`[3] contains the `.tar.gz` file for every official release including the `.git` tree.

`Repository`[4]
contains the git repository of the project.

[1] `http://www.ohwr.org/projects/fmc-adc-100m14b4cha`
[2] `http://www.ohwr.org/projects/fmc-adc-100m14b4cha-sw/documents`
[3] `http://www.ohwr.org/projects/fmc-adc-100m14b4cha-sw/files`
[4] `git://ohwr.org/fmc-projects/fmc-adc-100m14b4cha/fmc-adc-100m14b4cha-sw.git`

2.1 Use of Names in the Package

Within the code base you may find several different names to refer to the same card. Before you get confused we'd better explain the reasoning.

`fmc-adc, fmcadc`
: This is the most generic name, and the one that was used everywhere during development up to mid-June 2013. Unfortunately, we foresee several ADC cards can be used in the same computer, so the name was abandoned in favor of more specific names. The library header, however, is still called `fmcadc-lib.h` because the same API will be used for all such cards developed by our group.

`fmc-adc-100m14b4cha`
: This is the name of the `ohwr.org` project. It is the most specific name related to this project, but we use it rarely because it is cumbersome to type. Sometimes it is too long as well.

`fmc-adc-100m14b`
: The name of the kernel module. We think `fmc-` is important to keep because the driver fits in the *FMC* driver subsystem (which is included in the official kernel since version 3.11).

`adc-100m14b`
: The device name and the name of the default *gateware* file. The choice is dictated by the need for ZIO names to be 12 characters at most. We think the *fmc* part is irrelevant in this context, while keeping some feature details in case other ADC devices will coexist in the same host computer.

`fa_, zfad_, zfat_ (in the code)`
: Prefixes for functions and data, explained later. In the source code there is no need for symbols to be unique, and the role of prefixes is only so the reader to tell external names (like `printk` or `dma_ops`) from names that are defined within the project.

2.2 Names in the Repository

In the repository, the official releases are tagged with a date-based format `fmc-adc-sw-#yyyy-#mm` where *#yyyy* is the release year and *#mm* is the release month (e.g `fmc-adc-sw-2014-04`)

Within `ohwr.org`, we got the habit to include the package name in the tag name: before we started using git submodules, we used to tag several packages when a release was made, so we needed to tag with the package name. For this package we use submodules so we chose to tag with the "simple" name.

Note: If you got the code from the repository (as opposed to a named *tar.gz*) it may happen that you are building code from a later commit than what the manual claims to document. It is a fact of life that developers forget to re-read and fix documentation while updating the code. If work from the repository please use "`git describe HEAD`" and the appropriate *git log* command. We strongly suggest our user base to stick to official releases, though.

3 Driver Features

This driver is based on the *ZIO* framework and the *fmc-bus*. It supports initial setup of the board; it allows users to manually configure the board, to start and stop acquisitions, to force trigger, to read acquisition time-stamps and to read acquired samples.

4 Installation

This driver depends on two other drivers, as well as the Linux kernel. Also, it must talk to a specific FPGA binary file running in the carrier card (currently only the SPEC carrier is supported).

4.1 Gateware Dependencies

Please, refer to the ohwr wiki page[5] of the project to get the list of supported gateware for a specific release of the driver (and download them).

4.2 Gateware Installation

To install the FPGA image in the target system, you need to place the `.bin` file within '`/lib/firmware/`', where the system can find it.

By default, the driver looks for `fmc/adc-100m14b.bin`, so the full pathname is '`/lib/firmware/fmc/adc-100m14b.bin`'. You can find the copy of the last tested binary within the driver repository. Anyway, you can use a different name for your gateware file and you can load it by specifying the module parameter `file=`.

4.3 Software Dependencies

The kernel versions used during development are 3.2 and 3.6. Everything used here is known to at least compile with version 2.6.32,

The driver is base on the ZIO framework available on the Open Hardware Repository[6]. The version being used during development is branch *v1.0-fixes*, i.e. official release 1.0 with a few corrections and small changes. The exact commit being used is also used as a *git submodule* of this package, so it is automatically built.

The driver is also based on the *fmc-bus* available on the Open Hardware Repository[7]. This bus manages FMC carriers and mezzanines, identification and so on.

Both packages are currently checked out as *git submodules* of this package, and each of them is retrieved at the right version to be compatible with this driver. This means you may just ignore software dependencies and everything should work.

In February 2014 we also added *spec-sw* as a submodule, and later *svec-sw* too. While the carrier driver (i.e. the *device* object in a Linux bus) should not be needed to build the mezzanine driver (the *driver* in the bus), we think it's easier for our users. Also, we need the header because DMA operations are carrier-specific. The *svec-sw* submodule is not build by default: while we need to include its headers, it refers to some absolute pathnames that are internal to us, so it won't build for our external users.

4.4 Software Installation

To install this software package, you need to tell it where your kernel sources live, so the package can pick the right header files. You need to set only one environment variable:

`LINUX`

: The top-level directory of the Linux kernel you are compiling against. If not set, the default may work if you compile in the same host where you expect to run the driver.

[5] http://www.ohwr.org/projects/fmc-adc-100m14b4cha-sw/wiki/Releases

[6] http://www.ohwr.org/projects/zio

[7] http://www.ohwr.org/projects/fmc-bus

Additionally, to enable verbose debug messages, you can set `CONFIG_FMC_ADC_DEBUG=y` either in your environment or on the command line of *make*. Such messages are rarely useful for final users, unless they need to report a bug or other strange behaviour of the package.

Most likely, this is all you need to set. After this, you can run:

```
make
sudo make install LINUX=$LINUX
```

In addition to the normal installation procedure for `fmc-adc-100m14b.ko` you'll see the following message:

```
WARNING: Consider "make prereq_install"
```

The *prerequisite* packages are *zio* and *fmc-bus*; unless you already installed your own preferred version, you are expected to install the version this packages suggests. This step can be performed by:

```
make
sudo make prereq_install LINUX=$LINUX
```

The step is not performed by default to avoid overwriting some other versions of the drivers. After `make prereq_install`, the warning message won't be repeated any more if you change this driver and `make install` again.

In order to compile this package against a specific version of one of the related packages, you can use one or more of the following environment variables (again, this is mostly for developers):

`ZIO`

`FMC_BUS`

`SPEC_SW` The top-level directory of the repository checkouts for the pacakges. If unset, the top-level `Makefile` refers to the submodules of this package.

4.5 Module Parameters

The driver accepts a few load-time parameters for configuration. You can pass them to insmod directly, or write them in '`/etc/modules.conf`' or the proper file in '`/etc/modutils/`' .

The following parameters are used:

`file=/path/to/binary.bin`
: The binary file to use to reprogram the FPGA. The default value for this parameter is `fmc/adc-100m14b.bin` as seen in Section 4.2 [Gateware Installation], page 3. The name is a relative pathname from '`/lib/firmware`', and it will be used for each and every card.

`enable_test_data=1`
: This is for testing purpose. This option enables the testing data, so the ADC doesn't store samples, but fills memory with sequential numbers. The 64 bit data vector is filled with sequential values from a free-running 25 bit counter: channel 0 sweeps the full range, channel 1 goes from 0 to 511, other channel always report 0. Trigger detection is unaffected by use of test data.

`busid=NUMBER[,NUMBER...]`
: Restrict loading the driver to only a few mezzanine cards. If you have several SPEC cards, most likely not all of them host an ADC card; by specifying the list of bus identifiers you restrict the module to only drive those cards. This option will remain, but is going to be mostly obsoleted by use of eeprom-based identification of the cards.

5 About Source Code

5.1 Source Code Organization

The source code for the ADC driver and tools is split in three directories:

- '`kernel/`': this directory contains all source files to build the driver module. Each file represents a feature of the complete driver.
- '`lib/`': this directory contains all source files to build the users pace library.
- '`tools/`': this directory includes standalone tools that access the ADC driver directly. Their name begins with '`fau-`' which means *Fmc Adc User*. There is also a generic tool to generate pulses on the parallel port, so it has a different name pattern.
- '`libtools/`': this directory contains tools which use the fmcadc library. Their name begins with '`fald-`', which means *Fmc Adc Library Dependent*.

5.2 Source Code Conventions

This is a random list of conventions used in this package

- All the internal symbols used in the whole driver begin with the prefix `fa_`. The prefix mean: *Fmc Adc*.
- All internal symbols in the zio driver begin with `zfad_`. The prefix mean: *Zio Fmc Adc Driver*.
- All internal symbols in the zio trigger begin with `zfat_`. The prefix mean: *Zio Fmc Adc Trigger*.

6 Device Configuration

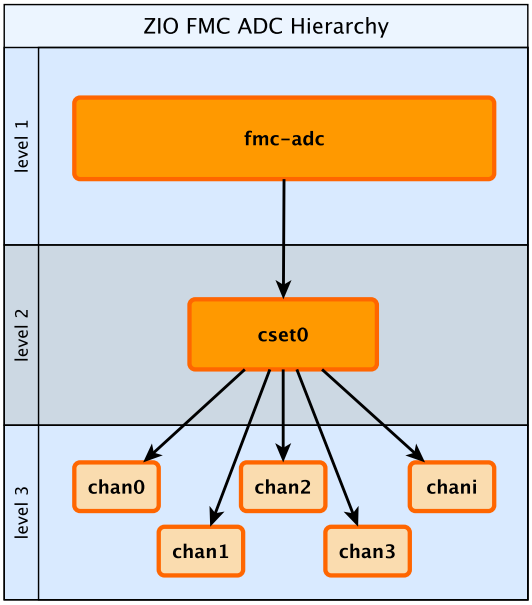
The driver is designed as a ZIO driver. ZIO is a framework for input/output hosted on http://www.ohwr.org/projects/zio.

ZIO devices are organized as *csets* (channel sets), and each of them includes channels. This device offers one cset and four channels. However, the device can only stores interleaved data for all four channels.

The current approach to this, implemented by one commit in thr *v1.0-fixes* branch of ZIO, is defining 5 channels: channels 0 to 3 are the actual input connectors, and their software counterpart is used to configure the channels; the last channel is called `i`, and is the *interleave* channel where data is retrieved.

6.1 The Overall Device

As said, the device has 1 cset with 4+1 channels. Channels from 0 to 3 represent che physical channels 1 to 4. The 5th channel *chani* represent a virtual channel created automatically by the ZIO framework; this channel represent the interleave acquisition on the cset.



The ADC registers can be accessed in the proper *sysfs* directory. For a card in slot 0 of bus 2 (like shown above), the directory is */sys/bus/zio/devices/adc-100m14b-0200*.

The overall device (*adc-100m14b*) doesn't offer configuration items besides its own temperature (read-only) because configuration is specific of the cset and the trigger, or the individual channel.

This is the content of the device-wide *sysfs* directory: it only includes standard attributes, the temperature and the cset subdirectory:

```
# ls -F /sys/bus/zio/devices/adc-100m14b-0200/
cset0/   devtype  enable  power/      temperature
devname  driver   name    subsystem   uevent
```

The temperature is reported as milli-degrees:

```
# cat /sys/bus/zio/devices/adc-100m14b-0200/temperature
51438
```

6.2 The Channel Set

The ADC has 1 Channel Set named '`cset0`'. Its attributes are used to control the ADC state machine, the channel parameters and so on.

This is the complete list of files in the respective *sysfs* directory.

```
# ls -F /sys/bus/zio/devices/adc-100m14b-0200/cset0/
ch0-50ohm-term  ch3-50ohm-term  devtype            tstamp-acq-end-b
ch0-offset      ch3-offset      enable             tstamp-acq-end-s
ch0-saturation  ch3-saturation  fsm-auto-start     tstamp-acq-end-t
ch0-vref        ch3-vref        fsm-command        tstamp-acq-stp-b
ch1-50ohm-term  chan0/          fsm-state          tstamp-acq-stp-s
ch1-offset      chan1/          name               tstamp-acq-stp-t
ch1-saturation  chan2/          power/             tstamp-acq-str-b
ch1-vref        chan3/          resolution-bits    tstamp-acq-str-s
ch2-50ohm-term  chani/          rst-ch-offset      tstamp-acq-str-t
ch2-offset      current_buffer  sample-decimation  tstamp-base-s
ch2-saturation  current_trigger sample-frequency   tstamp-base-t
ch2-vref        devname         trigger/           uevent
```

Some attributes are channel-specific, and one may thing they should live at channel-level. Unfortunately, ZIO currently lacks the mechanisms to convey channel attributes in the meta-data associated with an interleaved acquisition (where several channels coexist), and for this reason we chose to put them all at cset level. This may change in future releases, but the library implementation will follow, so there will be no effect on API users.

The description of attributes that follows is mainly useful for the shell user, to diagnose the system and hack around with parameters.

6.2.1 Channel-specific Cset Attributes

The *cset* includes three attributes for each channel, as follows:

`chN-50ohm-term`
: The read-write attribute accepts values 0 or 1. By writing 1, you turn on the termination resistor. Default is 0.

`chN-offset`
: The user offset is an integer value in the range [-5000,5000], and it represents millivolts. The offset represents the center-scale of conversion for the input channel. Internally, a DAC is used to generate the requested voltage, which is then subtracted from the input signal. DAC values are corrected according to the calibration values retrieved from the FMC EEPROM. For this reason, the offset may saturate at values less than +/- 5V.

`chN-vref`
: The "voltage reference" used for conversion. This attribute may be renamed to "range" in the future (again, with no effect on API users). Changing the range does not reset the user offset, which is automatically adjusted according to the new calibration values. The attribute accepts three values: 35 represents the 100mV range (-50mV to +50mV); 17 represents 1V range; 69 represents 10V range (-5V to +5V); 0 detaches the input connector from the ADC. The numbers used here derive from hardware values, and the attributes refuses any other value.

`cnN-saturation`
: The user saturation level in the range [0, 32767]. Users can use this value to configure their own saturation level. The hardware applies this value symmetrically on the negative side. By default is setted at the maximum value.

6.2.2 Generic Cset Attributes

This section lists the attributes that are defined by this driver; ZIO-wide attributes (`current_buffer`, `enable` and so on) are not described.

`fsm-auto-start`
: This attribute can be set to 1 or 0. It is 0 by default. If set to 1, the acquisition state-machine is automatically restarted after the previous run is complete. Thus, for example, a card configured for external trigger, after the first acquisition will continue aquiring and storing blocks to the ZIO buffer every time a new trigger event is detected. Applications can read such blocks from the char device.

`fsm-command`
: Write-only: start (1) or stop (2) the state machine. The values used reflects the hardware registers. Stopping the state machine aborts any ongoing acquisition. Starting the state machine is required in order to run an acquisition (the library manages this internally). The green LED *ACQ* on the front panel reflect the fact that the state machine has started. Restarting a running state machine is equivalent to first stopping it.

`fsm-state`
: Read-only current state of the FSM. Useful for diagnostics in strange situation. Please refer to the firmware manual (or to source code) about the various states.

`resolution-bits`
: This read-only attribute returns 14, the number of valid bits in the ADC data stream.

`rst-ch-offset`
: This write-only attributes zeroes all offset DACs when written, independently of the value being written. The driver applies the current calibration values, instead of writing 0 directly to the hardware.

`sample-decimation`
: The ADC always acquires at 100MSamples/s and this value cannot be changed (it actually can, but it is not currently supported nor even tested). If you need less samples you can tell the card to decimate (or under-sample) the data stream. The attribute accepts an integer value, 1 to 65536; it means to pick one sample every that many. Thus, but writing 100 you get a 1Ms data stream, nad by writing 2 you get a 50Ms data stream.

`sample-frequency`
: This read-only attributes returns the measured sampling frequency

6.2.3 Timestamp Cset Attributes

The ADC mark with a timestamp all these events: state machine start, state machine stop and acquisition end. The device split each timestamp in 3 attributes named: second (s), ticks (t) and bins (b).

Seconds represents (by default) the number of second since the epoch; ticks is the number of clocks at 125Mhz, the value is between 0 and 125000000 and it increments seconds when it overflow. At the moment, the bins register is unused.

For example, to read the entire timestamp of the state machine start event you should do:

```
# cat /sys/bus/zio/devices/adc-100m14b-0200/cset0/tstamp-acq-str-s
# cat /sys/bus/zio/devices/adc-100m14b-0200/cset0/tstamp-acq-str-t
# cat /sys/bus/zio/devices/adc-100m14b-0200/cset0/tstamp-acq-str-b
```

The driver export 4 time stamps:

`tstamp-acq-str-{s|t|b}`
: this is the time stamp of the last acquisition start command execution

`tstamp-acq-end-{s|t|b}`
: it is the time of last sample acquired

`tstamp-acq-stop-{s|t|b}`
: this is the time stamp of the last acquisition stop command execution

`tstamp-trg-lst-{s|t|b}`
: this is the time stamp of the last trigger fire. Please bear in mind that in multi-shot acquisition you have several trigger fire, so this time stamp refers only to the last one. If you need the time stamp for each trigger fire you have to get it from the zio_control of the associated acquisition block.

By default these time stamps represent (more or less) the time since the epoch. The user can change this and configure a different timing base. The attributes `tstamp-base-s` and `tstamp-base-t` are ment for this purpose.

6.3 The Channels

The ADC has 4 input channels. Each channel features one attribute, which reports the last acquired sample: `current-value`.

```
# ls -F /sys/bus/zio/devices/adc-100m14b-0200/cset0/chan0/
address  buffer/          current-value  devtype  name    uevent
alarms   current-control  devname        enable   power/
```

the current value is a 16 bit number, resulting from the 14 bit ADC value and calibration correction. The value is reported as unsigned, even if it actually represents a signed 16-bit integer. (This because ZIO manages 32-bit attributes and the value shown comes directly from the hardware).

```
# grep . chan*/current-value
chan0/current-value:1588
chan1/current-value:65436
chan2/current-value:384
chan3/current-value:644
```

Other attributes in the directory are defined by the kernel or by ZIO.

7 The Trigger

In ZIO, the trigger is a separate software module, that can be replaced at run time. This driver includes its own ZIO trigger type, that is selected by default when the driver is initialized. You can change trigger type (for example use the `timer` ZIO trigger) but this is not the typical use case for this board.

The name of the ADC trigger is `adc-100m14b`. Like all other ZIO objects, each instance of the trigger has a *sysfs* directory with its own attributes:

The ADC has its own *zio_trigger_type* and it can not work with any other ZIO's trigger. The ADC trigger is called **fmc-adc-trg**. We advise you against replacing the trigger with another one. The sysfs directory of this trigger is the following:

```
# ls -fF /sys/bus/zio/devices/adc-100m14b-0200/cset0/trigger/
delay     int-channel    polarity      sw-trg-enable     tstamp-trg-lst-t
devtype   int-threshold  post-samples  sw-trg-fire       uevent
enable    name           power/        tstamp-trg-lst-b
external  nshots         pre-samples   tstamp-trg-lst-s
```

The trigger supports three operating modes: the *external* trigger is driven by a specific LEMO connector on the front panel of the card. The *internal* trigger activates on data threshold in one of the four input channels – either positive-going or negative-going. The *software* trigger is activated by simply writing to a register.

This is the list of attributes (excluding kernel-generic and ZIO-generic ones):

`delay`

The delay attribute tells how many samples to delay actual acquisition since the trigger fired. Being sample-based, the resolution is 10ns. This applies to all trigger operating modes. By default `delay` is 0.

`enable`

This is a standard zio attribute, and the code uses it to enable or disable the hardware trigger (i.e. *internal* and *external*). By default the trigger is enabled.

`external`

The attribute is used to select the *internal trigger* (0) or the *external trigger* (1), within the realm of hardware modes.

`int-channel`
`int-threshold`

If the internal trigger is selected, these attributes choose the channel being monitored (range is 0..3) and the value of the data thresold (as a signed 16-bit value).

`nshots`

Number of trigger shots. The state machine acquires all trigger events to internal on-board memory, and performs DMA only at the end. In single-shot, the acquisition can be as long ad 32Msamples (on-board memory is 256MB), but in multi-shot acquisition is first done to in-FPGA memory, and thus each shot can only acquire 2048 samples.

`polarity`

Polarity for the data-thresold used in the internal trigger. 0 represents a positive-going signal (default), 1 represents a negative edge/slope.

`post-samples`
`pre-samples`

Number of samples to acquire. The pre-samples are acquired before the actual trigger event (plus its optional delay). The post samples start from the trigger-sample itself. The total number of samples acquired corresponds to the sum of the two numbers. For multi-shot acquisition, each shot acquires that many sample, but pre + post must be at most 2048.

`sw-trg-enable`
`sw-trg-fire`

To use the software trigger, you must first enable it (writing 1) to `sw-trg-enable`. When enabled, by writing any values to `sw-trg-file` you can force a trigger event. This is expected to be used only for diagnostic reasons.

`tstamp-trg-lst-b`
`tstamp-trg-lst-s`
`tstamp-trg-lst-t`

To be verified and documented.

8 The Buffer

In ZIO, buffers are separate objects. The framework offers two buffer types: *kmalloc* and *vmalloc*. The former uses the `kmalloc` function to allocate each block, the latter uses `vmalloc` to allocate the whole data area. While the *kmalloc* buffer is linked with the core ZIO kernel module, *vmalloc* is a separate module. The driver currently prefers *kmalloc*, but even when it preferred *vmalloc* (up to mid June 2013), if the respective module wad not loaded, ZIO would instantiate *kmalloc*.

You can change the buffer type, while not acquiring, by writing its name to the proper attribute. For example:

```
# echo vmalloc > /sys/bus/zio/devices/adc-100m14b-0200/cset0/current_buffer
```

The disadvantage of *kmalloc* is that each block is limited in size. usually 128kB (but current kernels allows up to 4MB blocks). The bigger the block the more likely allocation fails. If you make a multi-shot acquisition you need to ensure the buffer can fit enough blocks, and the buffer size is defined for each buffer instance, i.e. for each channel. In this case we acquire only from the interleaved channel, so before making a 1000-long multishot acquisition you can do:

```
# DEV=/sys/bus/zio/devices/adc-100m14b-0200
# echo 1000 > $DEV/cset0/chani/buffer/max-buffer-len
```

The *vmalloc* buffer allows *mmap* support, so when using *vmalloc* you can save a copy of your data (actually, you save it automatically if you use the library calls to allocate and fill the user-space buffer). However, a *vmalloc* buffer allocates the whole data space at the beginning, which may be unsuitable if you have several cards and acquire from one of them at a time.

The *vmalloc* buffer type starts off with a size of 128kB, but you can change it (while not aquiring), by writing to the associated attribute of the interleaved channel. For example this sets it to 10MB:

```
# DEV=/sys/bus/zio/devices/adc-100m14b-0200
# echo 10000 > $DEV/cset0/chani/buffer/max-buffer-kb
```

9 Summary of Attributes

The following table lists all attributes related to this driver. All values are 32-bit that ZIO framework can handle only 32bit unsigned integer.

Ctx	Name	RW	Def.	Accepted	Comment
Cset	enable	rw	1	[0;1]	
Cset	chN-50ohm-term	rw	0	[0;1]	N = 0..3
Cset	chN-offset	rw	0	[-5000; 5000]	mV, N = 0..3
Cset	chN-vref	rw	17	[0, 17, 35, 69]	N = 0..3
Cset	chN-saturation	rw	32767	[0;32767]	
Cset	fsm-auto-start	rw	0	[0;1]	
Cset	fsm-command	wo	-	[1;2]	2 = STOP
Cset	fsm-state	ro	-	-	hw values
Cset	resolution-bits	ro	14	-	
Cset	rst-ch-offset	wo	-	any	
Cset	sample-decimation	rw	1	[1;65535]	
Cset	sample-frequency	ro	-	-	
Cset	tstamp-base-s	rw	-	-	
Cset	tstamp-base-t	rw	-	-	
Cset	tstamp-acq-str-s	ro	-	-	
Cset	tstamp-acq-str-t	ro	-	-	
Cset	tstamp-acq-str-b	ro	-	-	
Cset	tstamp-acq-stp-s	ro	-	-	
Cset	tstamp-acq-stp-t	ro	-	-	
Cset	tstamp-acq-stp-b	ro	-	-	
Cset	tstamp-acq-end-s	ro	-	-	
Cset	tstamp-acq-end-t	ro	-	-	
Cset	tstamp-acq-end-b	ro	-	-	
Chan	current-value	ro	-	-	16 0..64k, use as signed 16b
Trig	delay	rw	0	[0;4G]	
Trig	enable	rw	1	[0;1]	enable hw trigger
Trig	external	rw	1	[0;1]	
Trig	int-channel	rw	0	[0;3]	
Trig	int-threshold	rw	0	[0;65535]	datum after offset/calib
Trig	nshots	rw	1	[0;65535]	
Trig	polarity	rw	0	[0;1]	1 = falling
Trig	post-samples	rw	0	Any	max 2k if multishot
Trig	pre-samples	rw	0	Any	max 2k if multishot
Trig	sw-trg-enable	rw	0	[0;1]	
Trig	sw-trg-fire	wo	-	Any	
Trig	tstamp-trg-s	ro	-	-	
Trig	tstamp-trg-t	ro	-	-	

Trig `tstamp-trg-b` ro - -

10 Reading Data with Char Devices

To read data from user-space, applications should use the ZIO char device interface. ZIO creates 2 char devices for each channel (as documented in ZIO documentation). The ADC acquires only interleaved samples, so ZIO creates two char device, as shown below:

```
# ls -l /dev/zio/
total 0
crw------- 1 root root 250, 8 Aug 23 22:21 adc-100m14b-0200-0-i-ctrl
crw------- 1 root root 250, 9 Aug 23 22:21 adc-100m14b-0200-0-i-data
```

The actual pathnames depend on the version of *udev* you are running. The *fmc-adc* library tries both names (the new one shown above, and the older one, without a `zio` subdirectory). Also, please note that a still-newer version of *udev* obeys device permissions, so you'll have read-only and write-only device files (in this case they are both read-only).

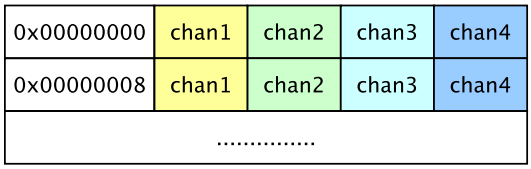
If more than one board is probed for, you'll have two or more similar pairs of devices, differing in the *dev_id* field, i.e. the `0200` shown above. The *dev_id* field is built using the PCI bus and the *devfn* octet; the example above refers to slot 0 of bus 2. (Most of the time each PCI-E physical slot is mapped as a bus, so the slot number is usually zero).

The ADC hardware does not allow to read data from a specific channel; data is only transferred as an interleaved block of samples. Neither the ZIO core nor the driver split interleaved data into 4 different buffers, because that task is computationally intensive, and is better left to the application (which may or may not need to do it). Thus, the driver returns to user-space a block of interleaved samples.

To read this interleaved block you can read directly the interleaved data char device *adc-100m14b-0200-0-i-data* using any program, for example *cat* or *hexdump*:

```
# hexdump -n 8 -e '"" 1/2 "%x\n"' /dev/zio/adc-100m14b-0200-0-i-data
fffc
e474
8034
8084
```

The ADC hardware always interleaves all 4 channels, and you cannot acquire a subset of the channels. The acquired stream, thus, follows this format:



The char-device model of ZIO is documented in the ZIO manual; basically, the `ctrl` device returns metadata dna thr `data` device returns data. Items in there are strictly ordered, so you can read metadata and then the associated data, or read only data blocks and discard the associated metadata.

The *zio-dump* tool, part of the ZIO distribution, turns metadata and data into a meaningful grep-friendly text stream.

11 Tools

The driver is distributed with a few tools. Some of them live in the '`tools/`' subdirectory, and some other are based on the provided library and live '`libtools/`' directory.

This chapter describes the former group; the tools' names use `fau-` as a prefix, for *Fmc Adc User*. For *libtools* see Chapter 13 [Library-based Tools], page 15.

11.1 Trigger Configuration

The program **fau-trg-config** configures the FMC ADC trigger. The tool offers command line parameters to configure every register exported by the driver. The help screen for the program summarizes the options:

```
# ./tools/fau-trg-config --help

fau-trg-config [OPTIONS] <DEVICE>

  <DEVICE>: ZIO name of the device to use
  --pre|-p <value>: number of pre samples
  --post|-P <value>: number of pre samples
  --nshots|-n <value>: number of trigger shots
  --delay|-d <value>: set the ticks delay of the trigger
  --threshold|-t <value>: set internal trigger threshold
  --channel|-c <value>: select the internal channel as trigger
  --external: set to external trigger. The default is the internal trigger.
  --negative-edge: set internal trigger polarity to negative edge. The default
                   is positive edge.
  --enable-sw-trg: enable the software trigger. By default is disabled.
  --disable-hw-trg: disable the hardware trigger. By default is enabled
  --force: force all attribute to the program default
  --help|-h: show this help

NOTE: The software trigger works only if also hardware trigger is enabled
```

The tool gets the configuration values from the user and it writes them to the corresponding *sysfs* attributes for the specified device. or example, if you want to configure the board for the external trigger and 3 shots of 10 pre-samples and 100 post-samples, this is the associated command line:

```
# ./tools/fau-trg-config --external --pre 10 --post 100 --re-enable 2 \
        adc-100m14b-0200
```

As shown, the *nshot* parameter is passed as a number of re-enables, because the trigger is initially automatically enabled. This may change in the future, for better naming consistency with hardware documentation and across tools.

11.2 Gain Configuration

The program `tools/fau-config-if` is a simple graphic tool that allow to select offset and range for the four channels, activate termination and see the current value of each channel, every 500ms.

The program open one window for each detected card, and configures it by writing to *sysfs*. Such writes are also reported to *stdout* (in the terminal where you invoked the program), so you can easily copy the pathnames in your shell commands.

Figure 11.1 shows two instances of the tool, running on the same card with device_id 0x200 (your window decorations will be different, according to your choice of window manager or desktop environment). The first one (at the left) is running under Tk-8.5; the second one shows the graphic appearance of Tk-8.4 (and earlier versions). If you prefer the older one, run "`wish8.4 tools/fau-config-if`" instead of "`tools/fau-config-if`" (or set the previous version as default Tk interpreter).

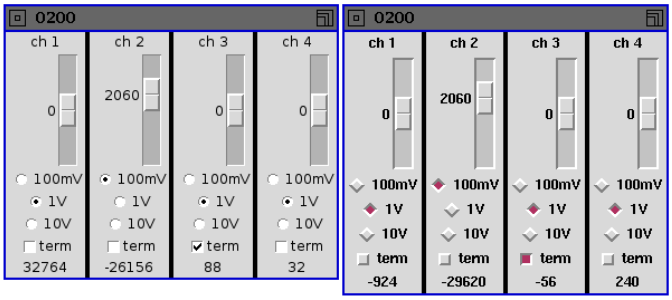


Figure 11.1

11.3 Acquisition Time

The program **fau-acq-time** retrieves the timestamps associated with the acquisition. This is the help screen of the program:

```
# ./tools/fau-acq-time --help

fau-acq-time [OPTIONS] <DEVICE>

  <DEVICE>: ZIO name of the device to use
  --last|-l : time between the last trigger and the acquisition end
  --full|-f : time between the acquisition start and the acquisition end
  --help|-h: show this help
```

The program can return two different "types" of acquisition time. The value returned by `--last` represent the time elapsed between the last trigger event and the acquire-end event; this is the time spent during the last capture.

The value returned by `--full` is the time elapsed between the acquisition start event and the acquisition end event, i.e. the total time spent waiting for all trigger events and the time spent acquiring all samples.

11.4 Parallel Port Burst

If you have a Parallel Port you can use it to generate bursts of pulses with a software program. This may be useful to test the external trigger; you can connect the parallel port to the external trigger of the FMC ADC and generate your trigger events with this program

The program *parport-burst*, part of this package, generates a burst according to three command line parameters: the I/O port of the data byte of the parallel port, the repeat count and the duration of each period. This example makes 1000 pulses of 100 usec each, using the physical address of my parallel port (if yours is part of the motherboard, the address is `378`):

```
# ./tools/parport-burst dd00 1000 100
```

12 The ADC Library

This package includes a library to access the ADC device through C language. The library is designed as a generic resource, that can be extended to support other devices, when they become available.

The library, also called API in some contexts, is described by a separate document, because over time we expect it to live in its own package.

13 Library-based Tools

In the directory '`libtools/`' you can find demo programs that use the official API to interact with the driver. The programs are meant to provide examples about use of the API. Unfortunately, the library is not yet finalized as we write this, but the tools in the package will remain in sync while we change the API to reach its final agreed-on implementation.

All library-based tools filenames use `fald-` as a prefix, for *Fmc Adc Library-Dependent*.

13.0.1 Simple Acquisition

The most important tool is '`fald-simple-acq`', which perform and ADC acquisition according to the specified parameters. The source code is meant to be used as an example by application authors, but sometimes the need for generality makes the code a slightly more difficult than expected. Also, a good amount of the program is devoted to command-line parsing, but that's unavoidable if we want to support several operating modes.

The program is invoked as "`fald-simple-acq <option> [...] <devid>`", where the *devid* is the hex identifier for the board (i.e., `200` or `0x200` for bus 2, slot 0).

This table describes the options supported (a few aliases are supported for backward compatibility, so old scripts still work, but such options are not documented); we support both short and long options:

`-b <num>, --before=<num>`
`-a <num>, --after=<num>`
: Number of pre-samples and post-samples for each "shot". If `nshots` is greater than 1, the total number of samples must be less than or equal to 2048. The total number of samples is the sum of pre-samples and post-samples.

`-n <num>, --nshots=<num>`
: Number of shots (i.e. number of trigger events to be served). For each shot, a complete acquisition is run. Each shot fills one ZIO data block, but no block is available before the card acquired the complete series.

`-d <num>, --delay=<num>`
: Delay from the trigger event to the actual use of the event. The delay is expressed in number of samples.

`-u <num>, --undersample=<num>`
: Undersample ("decimate") the acquired data. Only 1 sample every `<num>` is saved. The default is 1 (i.e., save all samples).

`-t <num>, --threshold=<num>`
`-c <chan>, --channel=<chan>`
: Select the "internal" trigger (i.e. data-driven). The channel number is expresses in the range 0..3; the threshold is in the range -32768..32767. By default the program selects the external trigger.

`--negative-edge`
: For either the external or internal trigger, use negative edges (by default: positive) to fire the trigger event.

`-B <file>, --binary=<file>`
`-M <basename>, --multi-binary=<basename>`
`-N, --dont-read`
: By default the program prints text data to standard output. When using one of these options, data is saved in binary format, or not read from the device at all. See below for a more complete description.

The program normally prints some diagnostic information about the acquisition to *stderr*, and the data, in a textual encapsulation to *stdout*.

For example, the following one is a 2-shot, 5-samples acquisition using the external trigger. The second channel is connected to the same electrical signal as the trigger connector.

```
spusa.root# ./libtools/fald-simple-acq -n 2 -b 2 -a 3 0x200
Read 20 samples from shot 1/2
   -2       -1449   -290   -872   -571
   -1       -2178   6325  -1175  -2110
    0       -1483  30745   -740  -2253
    1        -258  30745    201   -719
    2        -738  30745   -861  -2114
Read 20 samples from shot 2/2
   -2       -1052   -366   -234   1224
   -1       -1596   -309   -793   1186
    0       -2299   6431  -1303   -431
    1       -1555  30745   -680   -303
    2         -77  30745    261    979
```

By redirecting *stdout* to a file, you can easily plot the acquisition using *gnuplot* or other tools. In the example shown, we have two pre-samples (-2 and -1) and three post-samples (0, 1, 2).

By using `-B`, you tell the program to save both metadata and data to a binary file. Such a binary file can then be used offline, using tools like *zio-dump* (part of the ZIO distribution).

```
spusa.root# ./libtools/fald-simple-acq -n 2 -b 2 -a 3 -B /tmp/acq 0x200
Read 20 samples from shot 1/2
Read 20 samples from shot 2/2
spusa.root# ./zio/tools/zio-dump -c /tmp/acq
Ctrl: version 1.0, trigger adc-100m14b, dev adc-100m14b-0200, cset 0, chan 4
Ctrl: alarms 0x00 0x00
Ctrl: seq 5, n 20, size 2, bits 14, flags 01000011 (little-endian)
Ctrl: stamp 1372281727.026491000 (0)
Data: 81 78 f4 0e 16 fb ef fe 81 78 19 78 22 fc a9 fc
Data: 81 78 19 78 33 01 47 03 81 78 19 78 4a fe 54 fe
Data: 81 78 19 78 0b fb 7a fe

Ctrl: version 1.0, trigger adc-100m14b, dev adc-100m14b-0200, cset 0, chan 4
Ctrl: alarms 0x00 0x00
Ctrl: seq 6, n 20, size 2, bits 14, flags 01000011 (little-endian)
Ctrl: stamp 1372281727.026497354 (0)
Data: 39 fa 61 fe d8 fc fb fe 5d f8 00 20 ac fa 46 f8
Data: 41 fb 19 78 64 fd f8 f8 54 ff 19 78 c8 ff 27 fe
Data: 84 fd 19 78 e6 fb 89 f7
```

Zio dump can also show attributes, but it currently is unable to demultiplex an interleaved channel.

By using the *multi-binary* option, you can tell the program to save one file for each metadata and one file for each data. For example, lets' use the internal trigger on channel 2 and see the metadata for the first block:

```
spusa.root#  ./libtools/fald-simple-acq -n 2 -b 2 -a 3 -c 1 -t 10000 \
                     -M /tmp/multi 0x200
Read 20 samples from shot 1/2
Read 20 samples from shot 2/2
spusa.root# ls -l /tmp/multi*
-rw-r--r-- 1 root root 512 Jun 26 23:26 /tmp/multi.000.ctrl
-rw-r--r-- 1 root root  40 Jun 26 23:26 /tmp/multi.000.data
-rw-r--r-- 1 root root 512 Jun 26 23:26 /tmp/multi.001.ctrl
-rw-r--r-- 1 root root  40 Jun 26 23:26 /tmp/multi.001.data
spusa.root# ./adc/zio/tools/zio-dump -c -a /tmp/multi.000.ctrl
Ctrl: version 1.0, trigger adc-100m14b, dev adc-100m14b-0200, cset 0, chan 4
Ctrl: alarms 0x00 0x00
Ctrl: seq 7, n 20, size 2, bits 14, flags 01000011 (little-endian)
Ctrl: stamp 1372281961.064656080 (0)
Ctrl: device-std-mask: 0x0001
```

```
Ctrl: device-std-0   0x0000000e        14
Ctrl: device-ext-mask: 0x1fff
Ctrl: device-ext-0   0x00000001         1
[...]
Ctrl: trigger-std-mask: 0x0007
Ctrl: trigger-std-0   0x00000002         2
Ctrl: trigger-std-1   0x00000003         3
Ctrl: trigger-std-2   0x00000002         2
Ctrl: trigger-ext-mask: 0x001f
Ctrl: trigger-ext-0   0x00000000         0
Ctrl: trigger-ext-1   0x00000000         0
Ctrl: trigger-ext-2   0x00000001         1
Ctrl: trigger-ext-3   0x00002710     10000
Ctrl: trigger-ext-4   0x00000000         0
```

As shown, all acquisition parameters are part of the metadata ("control") file, as either standard or extended attributes. The mapping of standard attributes is defined at ZIO level (trigger attributes 1, 2, 3 are nshots, post-samples and pre-samples, resp.); extended attributes are defined by each device or trigger type (here, trigger attribute 3 is the threshold). The mapping of standard attributes is defined in the ZIO headers, the mapping of extended attributes is defined by each device or trigger.

Finally, by passing `-N`, you tell the program to not read data at all from the char devices. This allows reading directly with *zio-dump* or other programs, while still using *fald-simple-acq* to configure the acquisition.

13.0.2 Test Program

The program called `fald-test` is a very simple test program, that can allocate several buffers and fill them all, or use a single buffer over and over for multi-shot acquisition.

It is not documented for lack of time, but the source is meant to be readable. We used it and *strace* to check that stuff happens properly as documented. Configuration is done mainly by setting environment variables.

13.0.3 Retrieve Configuration

To retrieve the current configuration from a device you can use the demo program '`fald-simple-get-conf`':

```
spusa.root# ./adc/libtools/fald-simple-get-conf 0x200
Open ADC fmcadc_100MS_4ch_14bit dev_id 0x0200 ...
Get Trigger Configuration ...
    source: internal
    channel: 1
    threshold: 10000
    polarity: 0
    delay: 0
Get Acquisition Configuration ...
    n-shots: 2
    post-sample: 3
    pre-sample: 2
    decimation: 1
    frequency: 100000000Hz
    n-bits: 14
```

14 Troubleshooting

This chapter lists a few errors that may happen and how to deal with them.

14.1 ZIO Doesn't Compile

Compilation of ZIO ma fail with error like:

```
zio-ad788x.c:180: error: implicit declaration of function "spi_async_locked"
```

This happens because the function wasn't there in your older kernel version, and your system is configured to enable `CONFIG_SPI`.

To fix, please just remove the *zio-ad788x* line from `drivers/Makefile`.

14.2 make modules_install misbehaves

The command *sudo make modules_install* may place the modules in the wrong directory or fail with an error like:

```
make: *** /lib/modules/2.6.37+/build: No such file or directory.
```

This happens when you compiled by setting `LINUX=` and your *sudo* is not propagating the environment to its child processes. In this case, you should run this command instead

```
sudo make modules_install  LINUX=$LINUX
```