FMC TDC 1ns 5 Channel Documentation

Federico Vaga <federico.vaga@cern.ch>
Adam Wujek <dev_public@wujek.eu>

Jul 12, 2022

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

This document describes the gateware developed to support the FmcTDC 1n 5channel (later refered to as fmc-tdc) mezzanine card on the SPEC and SVEC carrier cards. The gateware is the HDL code used to generate the bitstream that configures the FPGA on the carrier (sometimes also called firmware). The gateware architecture is described in detail. The configuration and operation of the fmc-tdc is also explained. The Linux driver and basic tools are explained as well. On the other hand, this manual is not intended to provide information about the hardware design.

1.1 Repositories and Releases

The FMC TDC 1ns 5 Channels is hosted on the Open HardWare Repository. The main development happens here. You can clone the GIT project with the following command:

```
git clone https://ohwr.org/project/fmc-tdc.git
```

Within the GIT respository, releases are marked with a TAG named using the Semantic Versioning. For example the latest release is `v8.0.0`. You can also find older releases with a different versioning mechanism.

For each release we will publish the FPGA bitstream for all supported carrier cards (FPGA Bitstream Page). For the Linux driver we can’t release the binary because it depends on the Linux version on which it will run. For details about how to build the Linux driver for your kernel please have a look at *Compile And Install* section in *Driver’s Documentation*.

1.2 Documentation License

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/.

CHAPTER

TWO

HARDWARE DESCRIPTION

The *FmcTdc* is an FPGA Mezzanine Card (FMC - VITA 57 standard), containing a 5-channel Time To Digital Converter (TDC). All channels share same time base, therefore one can relate timestamps of pulses coming to different channels.

2.1 Requirements and Supported Platforms

FmcTdc can work with any VITA 57-compliant FMC carrier, provided that the carrier's FPGA has enough logic resources. This release of the driver software supports the following carriers:

- SPEC (Simple PCI-Express Carrier),
- SVEC (Simple VME64x Carrier)

In order to operate *FmcTdc*, the following hardware/software components are required:

- A standard PC with at least one free 4x (or wider) PCI-Express slot and a SPEC PCI-Express FMC carrier (supplied with an FmcTdc),
- In case of a VME version: any VME64x crate with a controller (tested on a MEN A20 and MEN A25) and a SVEC VME64x FMC carrier (supplied with one or two *FmcTdcs*),
- 50-ohm cables with 1-pin LEMO 00 plugs for connecting the I/O signals,
- Any Linux (kernel 3.10+) distribution.

2.2 Mechanical/Environmental

Mechanical and environmental specification:

- Format: FMC (VITA 57),
- Operating temperature range: 0 - 90 degC,
- Carrier connection: 160-pin Low Pin Count FMC connector.

Electrical Inputs/Outputs:

- 5 trigger inputs (LEMO 00),
- 6 LEDs: 5 for indicating input pulse, 1 as an PPS indicator,
- Carrier communication via 160-pin Low Pin Count FMC connector.

Trigger input:

- TTL/LVTTL levels, DC-coupled,

- 2 kOhm or 50 Ohm input impedance (software-selectable),
- Power-up input impedance: 2 kOhm,
- Protected against short circuit, overcurrent (> 200 mA) and overvoltage (up to +15 V),
- Maximum input pulse edge rise time: 20 ns.

Power supply:

- Used power supplies: P12V0, P3V3, P3V3 AUX, VADJ (voltage monitor only).

2.3 Timing

Input timing:

- Minimum pulse width: 100 ns. Pulses below 100 ns are rejected. Width checking is done in gateware by subtracting rising and falling edge timestamps.
- Minimum pulse spacing: 100 ns.
- Only rising edges are time tagged.

Time base:

- On-board oscillator accuracy: +/- 4 ppm (i.e. max. 4 ns error for pulses separated by 1 ms).
- When using White Rabbit as the timing reference: depending on the characteristics of the grandmaster clock and the carrier used. Usually < 1ns.

Timestamp transfer modes:

- DMA on SPEC-carrier
- FIFO on SVEC-carrier

Performance:

- TDC precision: 700 ps peak-peak (six sigma). Outliers of $\pm$4 ns are observed at the expected frequency of ~ 1 outlier/10M measurements.
- TDC resolution: 81 ps.
- Maximum input pulse rate: Transfer-mode and CPU dependent; some examples:
  - DMA-mode on SPEC-carrier, Siemens IPC847E : continuous 200KHz with the minimal processing of samples
  - DMA-mode on SPEC-carrier, Siemens IPC847E : 1 MHz (total from the 5 channels) in burst of 5k samples
  - FIFO-mode on SVEC-carrier, VME MENA-25: continuous 80KHz with the minimal processing of samples

THE GATEWARE

3.1 About Source Code

3.1.1 Build from Sources

The fmc-tdc hdl design make use of the `hdlmake` tool. It automatically fetches the required hdl cores and libraries. It also generates Makefiles for synthesis/par and simulation.

Here is the procedure to build the FPGA binary image from the hdl source.:

```
# Install ``hdlmake`` (version 3.4).
# Get fmc-tdc hdl sources.
git clone https://ohwr.org/project/fmc-tdc.git <src_dir>

# Goto the synthesis directory.
cd <src_dir>/hdl/syn/<carrier>/

# Fetch the dependencies and generate a synthesis Makefile.
hdlmake

# Perform synthesis, place, route and generate FPGA bitstream.
make
```

3.1.2 Source Code Organisation

hdl/rtl/
TDC specific hdl sources.

hdl/ip_cores/
Location of fetched hdl cores and libraries.

hdl/top/<design>
Top-level hdl module for selected design.

hdl/syn/<design>
Synthesis directory for selected design. This is where the synthesis top manifest, the design constraints and the ISE project are stored. For each release, the synthesis, place&route and timing reports are also saved here.

hdl/testbench/
Simulation files and testbenches.

3.1.3 Dependencies

The fmc-tdc gateware depends on the following hdl cores and libraries: General Cores, DDR3 SP6 core, GN4124 core (SPEC only), SPEC (SPEC only) VME64x Slave (SVEC only), SVEC (SVEC only), WR Cores.

These dependencies are managed with GIT submodules. Whenever you checkout a different branch remember to update the submodules as well.:

```
git submodule sync
git submodule update
```

3.2 Data Format

The TDC gateware is retrieving timestamps generated by the ACAM chip, it is adapting them to a comprehensive format and it is then making them available to the PCIe interface in a circular buffer. Each final timestamp is a 128-bit word with the following structure:

Bits	Description
[127:96]	Metadata
	[127..125]: Input Channel from 0 to 4
	[123] : Edge Type, "1" means rising edge, "0" means falling
	[122..96] : not used
[95:64]	Local UTC time: the resolution is 1 s
[63:32]	Coarse time within the current UTC time: the resolution is 8 ns
[31:0]	Fine time: the resolution is 81.03 ps

As the structure indicates, each timestamp is referred to a UTC second. The coarse and fine times indicate with 81.03 ps resolution the amount of time passed after the last UTC second.

CHAPTER

FOUR

THE SOFTWARE

4.1 Driver

4.1.1 Driver Features

4.1.2 Requirements

The fmc-tdc device driver has been developed and tested on Linux 3.10. Other Linux versions might work as well but it is not guaranteed.

This driver depends on the zio framework and fmc library; we developed and tested against version zio 1.4 and fmc 1.1.

The FPGA address space must be visible on the host system. This requires a driver for the FPGA carrier that exports the FPGA address space to the host. As of today we support SPEC and SVEC.

4.1.3 Compile And Install

The compile and install the fmctdc1ns5ch device driver you need first to export the path to its direct dependencies, and then you execute `make`. This driver depends on the zio framework and fmc library; on a VME system it depends also on the VME bridge driver from CERN BE-CEM. Additionally it is assumed that location of wbgen2 is available via `PATH` variable.

```
$ cd /path/to/fmc-tdc/software/kernel
$ export LINUX=/path/to/linux/sources
$ export ZIO=/path/to/zio
$ export FMC=/path/to/fmc-sw
$ export VMEBUS=/path/to/vmebridge
$ make
$ make install
```

Note: Since version v8.0.0 the fmc-tdc device driver does not depend anymore on fmc-bus subsystem, instead it uses a new fmc library

The building process generates 3 Linux modules: *kernel/fmc-tdc.ko*, *kernel/fmc-tdc-spec.ko* (for SPEC card), and *kernel/fmc-tdc-svec.ko* (for SVEC card).

4.1.4 Drivers' Dependencies

The TDC driver requires the following drivers to function:

- if the used carrier is SPEC then from spec repository: *gn412x-fcl.ko*, *gn412x-gpio.ko*, *spec-gn412x-dma.ko* and *spec-fmc-carrier.ko*
- if the used carrier is SVEC then *vmebus.ko*
- from general-cores repository: *spi-ocores.ko*, *i2c-ocores.ko* and *htvic.ko* (more details in the section *Building General Cores drivers*)
- from zio repository: *zio-buf-vmalloc.ko* and *zio.ko* (more details in the section *Building ZIO drivers*)
- from fmc repository: *fmc.ko* (more details in the section *Building FMC driver*)
- drivers from the kernel tree: *mtd.ko*, *at24.ko*, *m25p80.ko*, *i2c_mux.ko* and *fpga-mgr.ko* (available in kernels v4.4 and newer, for older kernels see section *Building FPGA manager driver*)

In addition the following tools are required to build above drivers:

- cheby (more details in the section *Installing Cheby*)
- wbgen2 (more details in the section *Installing Wbgen2*)

Please read the following subsections for details

4.1.5 Building Drivers

This subsection describes the build process of Linux Device Drivers used by the TDC and tools needed during their build.

Installing Cheby

Clone *cheby* repository:

```
$ git clone https://gitlab.cern.ch/cohtdrivers/cheby.git
```

Install cheby:

```
$ cd cheby
$ python setup.py install
```

It may be required to install *python-setuptools* or *python-setuptools.noarch* package using your Linux distribution's software manager.

Installing Wbgen2

Clone *wbgen2* repository:

```
$ git clone https://ohwr.org/project/wishbone-gen.git
```

If needed export the location of *wbgen2* (needed for *fmc-tdc* drivers compilation):

```
export WBGEN2=/path/to/wishbone-gen/wbgen2
```

Building FPGA Manager driver

If kernel module *fpga-mgr.ko* is not available in the kernel that is used, probably the backported version is needed.

Clone backported *fpga-manager* repository:

```
$ git clone https://gitlab.cern.ch/coht/fpga-manager.git
```

Build and install kernel module (*fpga-mgr.ko*):

```
$ cd fpga-manager
$ export LINUX=/path/to/linux/sources
$ make
$ make install
```

Building ZIO drivers

Clone *zio* repository:

```
$ git clone https://ohwr.org/misc/zio.git
```

Build and install kernel modules (*zio-buf-vmalloc.ko* and *zio.ko*):

```
$ cd zio
$ export LINUX=/path/to/linux/sources
$ make
$ make install
```

Building General cores drivers

Clone *general-cores* repository:

```
$ git clone https://ohwr.org/project/general-cores.git
```

Build and install kernel modules (*spi-ocores.ko*, *i2c-ocores.ko* and *htvic.ko*):

```
$ cd general-cores/software
$ export LINUX=/path/to/linux/sources
$ make
$ make install
```

Building FMC driver

Clone *fmc* repository:

```
$ git clone https://ohwr.org/project/fmc-sw.git
```

Build and install kernel module (*fmc.ko*):

> $ cd fmc-sw/ $ export LINUX=/path/to/linux/sources $ make $ make install

Building SPEC drivers

Clone *spec* repository:

```
$ git clone https://ohwr.org/project/spec.git
```

Build and install kernel modules (*gn412x-fcl.ko*, *gn412x-gpio.ko*, *spec-gn412x-dma.ko* and *spec-fmc-carrier.ko*):

```
$ cd spec/software
$ export CHEBY=/path/to/cheby/bin/cheby
$ export I2C=/path/to/general-cores/software/i2c-ocores
$ export SPI=/path/to/general-cores/software/spi-ocores
$ export FPGA_MGR=/path/to/fpga-manager
$ export FMC=/path/to/fmc-sw
$ export LINUX=/path/to/linux/sources
$ make
$ make install
```

Building SVEC drivers

Building missing mainline drivers

It may happen that your system lacks of drivers that are included into the mainline Linux kernel. This section describes how to build *i2c-mux.ko* and *m25p80.ko* drivers for CENTOS 7.

The first step is to download the Linux sources that mach the version used in your system and unpack them using your favorite method. Then prepare sources for a compilation:

::

make prepare

Select missing drivers by adding `CONFIG_I2C_MUX=m` and `CONFIG_MTD_M25P80=m` to .config manually, or with a favorite tool (like `menuconfig`. Start the build of missing drivers:

```
make M=drivers/i2c/
make M=drivers/mtd/devices/
```

Copy drivers from `drivers/mtd/devices/m25p80.ko` and `drivers/i2c/i2c-mux.ko` to a known place.

4.1.6 Top Level Driver

The fmc-tdc is a generic driver for an FPGA device that could be instanciated on a number of FMC carriers. For each carrier we write a little Linux module which acts as a top level driver (like the MFD drivers in the Linux kernel). In these modules there is the knowledge about the virtual memory range, the IRQ lines, and the DMA engine to be used.

The top level driver is a platform driver that matches a string containing the application identifier. The carrier driver builds this identification string from the device ID embedded into the FPGA (https://ohwr.org/project/fpga-dev-id).

4.1.7 Loading drivers for SPEC

Load drivers *at24.ko* and *mtd.ko*. They should be distributed with your Linux distribution in package like `kernel-plus` for CENTOS 7 of `linux-modules` for Ubuntu.

```
sudo modprobe at24
sudo modprobe mtd
```

Load drivers from the mainline Linux:

```
sudo insmod i2c-mux.ko
sudo insmod m25p80.ko
```

Load *fmc* drivers:

```
sudo insmod fmc.ko
```

Load *fpga-manager* drivers:

```
sudo insmod fpga-mgr.ko
```

Load drivers from *general-cores*:

```
sudo insmod htvic.ko
sudo insmod i2c-ocores.ko
sudo insmod spi-ocores.ko
```

Load drivers from *spec-sw*:

```
sudo insmod spec-gn412x-dma.ko
sudo insmod gn412x-gpio.ko
sudo insmod gn412x-fcl.ko
sudo insmod spec-fmc-carrier.ko
```

If you use the custom path to the firmware, set it at the latest at this point.

```
echo -n <path_to_bitstreams> | sudo tee /sys/module/firmware_class/parameters/path
```

Load bitstream into SPEC's FPGA:

```
echo -n <bitstream.bin> | sudo tee /sys/kernel/debug/<PCIe_device>/fpga_firmware
```

Load the ZIO and TDC drivers:

```
sudo insmod zio.ko
sudo insmod zio-buf-vmalloc.ko
sudo insmod fmc-tdc.ko
sudo insmod fmc-tdc-spec.ko
```

4.1.8 Loading drivers for SVEC

For SVEC the loading procedure is very similar to SPEC. It is required to load *svec-fmc-carrier.ko* and *fmc-tdc-svec.ko* instead of *spec-fmc-carrier.ko* and *fmc-tdc-spec.ko*. Additionally, there is no need to load *spec-gn412x-dma.ko*, *gn412x-gpio.ko* and *gn412x-fcl.ko*, since these drivers are specific to SPEC.

```
sudo modprobe at24
sudo modprobe mtd
sudo insmod i2c-mux.ko
sudo insmod m25p80.ko
sudo insmod fmc.ko
sudo insmod fpga-mgr.ko
sudo insmod htvic.ko
sudo insmod i2c-ocores.ko
sudo insmod spi-ocores.ko
sudo insmod svec-fmc-carrier.ko
echo -n <path_to_bitstreams> | sudo tee /sys/module/firmware_class/parameters/path
echo -n <bitstream.bin> | sudo tee /sys/kernel/debug/svec-vme.<slot>/fpga_firmware
sudo insmod zio.ko
sudo insmod zio-buf-vmalloc.ko
sudo insmod fmc-tdc.ko
sudo insmod fmc-tdc-svec.ko
```

4.1.9 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:

irq_timeout_ms=NUMBER
: It sets the IRQ coalesing timeout expressed in milli-seconds (ms). By default the value is set to 10ms.

test_data_period=NUMBER
: It sets how many fake timestamps to generate every seconds on the first channel, 0 to disable. By default the value is set to 0.

dma_buf_ddr_burst_size=NUMBER
: It sets DDR size coalesing timeout expressed in number of timestamps. By default the value is set to 16 timestamps.

wr_offset_fix=NUMBER
: It overwrites the White-Rabbit calibration offset for calibration value computed before 2018. By default this is set to 229460 ps.

4.1.10 Device Abstraction

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

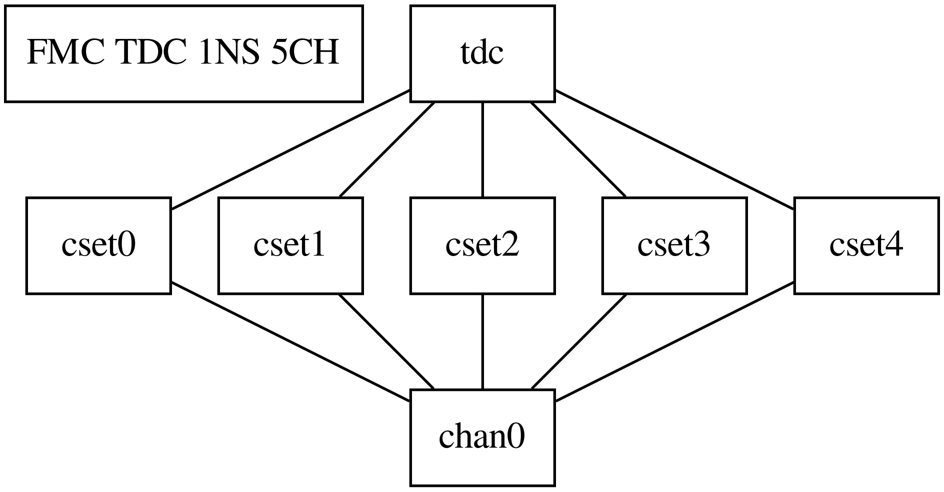
The driver is designed as a ZIO driver. ZIO is a framework for input/output hosted on http://www.ohwr.org/projects/zio.

ZIO devices are organized as csets (channel sets), and each of them includes channels. All channels belonging to the same cset trigger together. This device offers a channel-set for each channel.

Note: Unless specified, the units are the same as for the TDC HDL design. Therefore, this driver does not perform any data processing.

The Overall Device

As said, the device has 5 cset with 1 channel each. Channel sets from 0 to 4 represent the physical channels 1 to 5. In other words a channel set represents a single TDC channel.



The TDC registers can be accessed in the proper sysfs directory:

```
cd /sys/bus/zio/devices/tdc-1n5c-${ID}
```

The overall device (*tdc-1n5c*) provides the following attributes:

calibration_data
: It is a binary attribute which allows the user to change the run-time calibration data (the EEPROM will not be touched). The `fmc-tdc-calibration` tool can be used to read write calibration data. To be consistent, this binary interface expects **only** little endian values because this is the endianness used to store calibration data for this device.

coarse
: Coarse part of the current TAI time. This value is in nanoseconds with 8 ns resolution. The `fmc-tdc-time` tool can be used to read TAI time.

command
: Send the command to the driver. As today it is possible to enable/disable White Rabbit, set the board to the current time or check the source of the timing. The `fmc-tdc-time` tool can be used to send the commands related to the current time source.

seconds
: Current TAI time in seconds. The `fmc-tdc-time` tool can be used to read TAI time.

temperature
: It shows the current temperature. To get the temperature in C degrees use the formula `temperature/16`. The `fmc-tdc-temperature` tool can be used to read the temperature.

transfer-mode
: It shows the current transfer mode. 0 for FIFO, 1 for DMA.

wr-offset
: Offset used by White Rabbit.

The Channel Set

The TDC has 5 Channel Sets named `cset[0-4]`. Its attributes are used to control and monitor each TDC channel individually. All channel specific attributes are available at the channel set level.

The Channels

Because there is a one-to-one relation with the channel set, we have decided to put all custom attributes at the channel set level. So, at this level you will find only default ZIO attributes.

The Trigger

TODO fix this section

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.

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

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.

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.

The Buffer

TODO fix this section

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 was 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/tdc-1n5c-0004/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:

```
export DEV=/sys/bus/zio/devices/tdc-1n5c-0004
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 acquiring), by writing to the associated attribute of the interleaved channel. For example this sets it to 10MB:

```
export DEV=/sys/bus/zio/devices/tdc-1n5c-0004
echo 10000 > $DEV/cset0/chani/buffer/max-buffer-kb
```

4.1.11 The debugfs Interface

When the DMA mode is used, the fmctdc1ns5cha driver exports a set of debugfs attributes which are supposed to be used only for debugging activities. For each device instance you will see a directory in `/sys/kernel/debug/fmc-tdc.*`.

regs
  It dumps the FPGA registers

4.1.12 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 TDC can acquire data on each channel independently, so ZIO creates ten char device, as shown below:

```
$ ls -l /dev/zio/tdc-*
  cr--r----- 1 root root 241, 0 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-0-0-ctrl
  cr--r----- 1 root root 241, 1 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-0-0-data
  cr--r----- 1 root root 241, 2 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-1-0-ctrl
  cr--r----- 1 root root 241, 3 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-1-0-data
  cr--r----- 1 root root 241, 4 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-2-0-ctrl
```

(continues on next page)

(continued from previous page)

```
  cr--r----- 1 root root 241, 5 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-2-0-data
  cr--r----- 1 root root 241, 6 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-3-0-ctrl
  cr--r----- 1 root root 241, 7 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-3-0-data
  cr--r----- 1 root root 241, 8 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-4-0-ctrl
  cr--r----- 1 root root 241, 9 Jan 13 13:36 /dev/zio/tdc-1n5c-000b-4-0-data
```

If more than one board is probed for, you'll have more similar pairs of devices, differing in the dev_id field, i.e. the `000b` shown above. The dev_id field is assigned by the Linux kernel platform subsystem.

The char-device model of ZIO is documented in the ZIO manual; basically, the ctrl device returns metadata and the 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.

4.1.13 User Header Files

Both the kernel and the user make use of the same header file `fmc-tdc.h`. This because they need to share some data stracture and constants use to interpret data and meta-data in the library or by an application

Troubleshooting

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

Installation issue with modules_install

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

```
make: *** /lib/modules/<kernel-version>/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
```

4.2 Tools

The driver is distributed with a few tools living in the `tools/` subdirectory, most of these tools use the fmc-tdc library. The programs are meant to provide examples about the use of the driver and library interface.

4.2.1 List TDC boards

The tool `fmc-tdc-list` is capable of listing the available boards in the system. Below is the output from the command on an example system with 3 SPEC boards, each populated with a TDC mezzanine.

```
$ fmc-tdc-list
FMC-TDC Device ID 0019
FMC-TDC Device ID 0018
FMC-TDC Device ID 0017
```

4.2.2 Termination Configuration

The tool `fmc-tdc-term` enables or disables the 50 Ohm termination of a given input channel. The listing below shows the run of `fmc-tdc-term` tool to get the current status of the 50 Ohm termination on the TDC board with an ID assigned to 4:

```
$ fmc-tdc-term 0x4
channel 0: 50 Ohm termination is off
channel 1: 50 Ohm termination is off
channel 2: 50 Ohm termination is off
channel 3: 50 Ohm termination is off
channel 4: 50 Ohm termination is off
```

To set the 50 Ohm termination e.g. on channel 0 on the TDC board with an ID assigned to 4 please execute the following command:

```
$ fmc-tdc-term 0x4 0 on
channel 0: 50 Ohm termination is on
```

4.2.3 Reading Temperature

The tool `fmc-tdc-temperature` allows to read the current temperature of the TDC board. The command below reads the temperature of the TDC board with an ID assigned to 4:

```
$ fmc-tdc-temperature 0x4
31.4 deg C
```

4.2.4 Getting And Setting Board Time

The tool `fmc-tdc-time` allows to read and switch the time source to White-Rabbit or local oscillator. The command below gets the information about the current time source:

```
$ fmc-tdc-time 0x4 get
WR Status: synchronized.
Current TAI time is 1647471357.000000000 s
```

In the example above, the time source has been set to White-Rabbit. To set the time source to the local oscillator:

```
$ fmc-tdc-time 0x4 local
# no output after the command is executed
```

To set the time source to the White-Rabbit:

```
$ fmc-tdc-time 0x4 wr
Locking the card to WR: ... locked!
```

4.2.5 Read Timestamps

The tool `fmc-tdc-tstamp` can print acquired timestamps. In the example below the tool prints 5 samples (`-s` parameter) from the channel 2 (`-c` parameter) on the board with the ID 0x19 (`-D` parameter).

```
fmc-tdc-tstamp -D 0x19 -c 2 -s 5
channel 2 | channel seq 0
    ts   0000041028s  590492339195ps
    diff 0000041028s  590492339195ps [0.000024 Hz]
channel 2 | channel seq 1
    ts   0000041028s  591492339023ps
    diff 0000000000s  000999999828ps [1000.001000 Hz]
channel 2 | channel seq 2
    ts   0000041028s  592492338931ps
    diff 0000000000s  000999999908ps [1000.001000 Hz]
channel 2 | channel seq 3
    ts   0000041028s  593492338597ps
    diff 0000000000s  000999999666ps [1000.001000 Hz]
channel 2 | channel seq 4
    ts   0000041028s  594492338425ps
    diff 0000000000s  000999999828ps [1000.001000 Hz]
```

4.2.6 User Offset Configuration

The tool `fmc-tdc-offset` sets or gets the user-offset applied to the incoming timestamps. The example below show that all offsets are set to 0 in an example setup.

```
$ fmc-tdc-offset 0x19
channel 0: 0 ps
channel 1: 0 ps
channel 2: 0 ps
channel 3: 0 ps
channel 4: 0 ps
```

4.2.7 Calibration Data

The tool `fmc-tdc-calibration` reads calibration data from a file that contains it in binary form and shows it on STDOUT in binary form or in human readable one (default). This could be used to change the TDC calibration data at runtime by redirecting the binary output of this program to the proper sysfs binary attribute. This tool expects all values to be little endian. Please note that the TDC driver supports only ps precision, but calibration data is typically stored with sub-picosecond precision. For this reason, according to your source, calibration values may disagree on the fs part.

The example below shows the read of calibration data:

```
$ fmc-tdc-calibration -f /sys/bus/zio/devices/tdc-1n5c-0004/calibration_data
Temperature: 47 C
White Rabbit Offset: 229460000 fs
Zero Offset
  ch1-ch2: -109000 fs
  ch2-ch3: 493000 fs
  ch3-ch4: 499000 fs
  ch4-ch5: 336000 fs
```

4.3 The Library

Here you can find all the information about the *fmc-tdc* API and the main library behaviour that you need to be aware of to write applications.

This document introduces the developers to the development with the TDC library. Here you can find an overview about the API, the rational behind it and examples of its usage. It is not the purpose of the document to describe the API details. The complete API is available in *the Library API* section.

Note: The TDC hardware design diverged into different buffering structures. One based on FIFOs for SVEC, and one based on double-buffering in DDR for SPEC. The API tries to provide the same user-experience, however this is not always possible. Functions having different behaviour are properly declaring it in their documentation.

Note: This document provides also snippet of code from *example.c*. This is only to show you an example, please avoid to blindly copy and paste.

4.3.1 Initialization and Cleanup

The library may keep internal information, so the application should call its initialization function `fmctdc_init()`. After use, it should call the exit function `fmctdc_exit()` to release any internal data.

Note: `fmctdc_exit()` is not mandatory, the operating system releases anything in any case – the library doesn't leave unexpected files in persistent storage.

These functions don't do anything at this point, but they may be implemented in later releases. For example, the library may scan the system and cache the list of peripheral cards found, to make later *open* calls faster. For this reason it is **recommended** to, at least, initialize and release the library before starting.

Following an example from the `example.c` code available under `tools`

```
        err = fmctdc_init();
        if (err)
                exit(EXIT_FAILURE);

        err = use_fmctdc_library();
        if (err)
                exit(EXIT_FAILURE);
```

(continues on next page)

(continued from previous page)

```
        fmctdc_exit(); /* optional, indeed in the error condition
                          we do not do it */
```

4.3.2 Error Reporting

Each library function returns values according to standard *libc* conventions: -1 or NULL (for functions returning `int` or pointers, resp.) is an error indication. When error happens, the *`errno`* variable is set appropriately.

The *`errno`* values can be standard Posix items like `EINVAL`, or library-specific values, for example `FMCTDC_ERR_VMALLOC` (*driver vmalloc allocator not available*). All library-specific error values have a value greater than 4096, to prevent collision with standard values. To convert such values to a string please use *`fmctdc_strerror()`*

Following an example from the `example.c` code available under `tools`

```
                fprintf(stderr, "%s: Cannot open device: %s\n",
                        prog_name, fmctdc_strerror(errno));
```

4.3.3 Opening and closing

Each device must be opened before use by calling *`fmctdc_open()`*, and it should be closed after use by calling *`fmctdc_close()`*.

Note: *`fmctdc_close()`* is not mandatory, but it is recommended, to close if the process is going to terminate, as the library has no persistent storage to clean up – but there may be persistent buffer storage allocated, and *`fmctdc_close()`* may release it in future versions.

The data structure returned by *`fmctdc_open()`* is an opaque pointer used as token to access the API functions. The user is not supposed to use or modify this pointer.

Another kind of open function has been provided to satisfy CERN's developers needs. Function *`fmctdc_open_by_lun()`* is the open by LUN (*Logic Unit Number*); here the LUN concept reflects the *CERN* one. The usage is exactly the same as *`fmctdc_open()`* only that it uses the LUN instead of the device ID.

No automatic action is taken by *`fmctdc_open()`*. Hence, you may want to flush the buffers before starting a new acquisition session. You can do this with *`fmctdc_flush()`*

```
        tdc = fmctdc_open(0x0000);
        if (!tdc) {
                fprintf(stderr, "%s: Cannot open device: %s\n",
                        prog_name, fmctdc_strerror(errno));
                return -1;
        }

        err = fmctdc_flush(tdc, channel);
        if (err)
                return err;

        err = config_and_acquire(tdc);
        if (err) {
                fprintf(stderr, "%s: Error: %s\n",
                        prog_name, fmctdc_strerror(errno));
```

(continues on next page)

(continued from previous page)

```
                return -1;
        }

        fmctdc_close(tdc);
```

4.3.4 Configuration and Status

The TDC configuration API is based on a number of getter and setter function for each option. These include: *termination*, *IRQ coalescing timeout*, *board time*, *white-rabbit*, *timestamp mode*.

The *termination* options allows you to set the 50 Ohm channel termination. You can use the following getter and setter: *fmctdc_get_termination()*, *fmctdc_set_termination()*.

```
        err = fmctdc_set_termination(tdc, channel, termination);
        if (err)
                return err;
        termination_rb = fmctdc_get_termination(tdc, channel);
        if (termination_rb < 0)
                return termination_rb;
```

The *IRQ coalescing timeout* option allows to force an IRQ when the timeout expire to inform the driver that there is at least one pending timestamp to be transfered. You can use the following getter and setter: *fmctdc_coalescing_timeout_get()*, *fmctdc_coalescing_timeout_set()*.

```
        err = fmctdc_coalescing_timeout_set(tdc, channel, coalescing_timeout);
        if (err)
                return err;
        err = fmctdc_coalescing_timeout_get(tdc, channel, &coalescing_timeout_rb);
        if (err)
                return err;
```

The TDC main functionality is to timestap incoming pulses. To assign a timestamp the board needs a time reference. This can be provided by the on-board clock, or by the more accurate white-rabbit network. You can enable or disable white-rabbit using *fmctdc_wr_mode()*. You can check the white-rabbit status with *fmctdc_check_wr_mode()*. When working with white-rabbit the time reference is handled by the white-rabbit network.

```
        err = fmctdc_wr_mode(tdc, wr_mode);
        if (err)
                return err;
        wr_mode_rb = fmctdc_check_wr_mode(tdc);
        if (wr_mode_rb < 0)
                return wr_mode_rb;
```

If you do not have white-rabbit connected to the TDC, or simply this is not what you want, then be sure to disable. When white-rabbit is disabled the TDC will use the on-board clock to keep a time reference. However, in this scenario the user is asked to set first the time using *fmctdc_set_time()* or *fmctdc_set_host_time()*.

```
        err = fmctdc_set_time(tdc, &time);
        if (err)
                return err;
```

Whater you are using white-rabbit or not, you can get the current board time with *fmctdc_get_time()*.

```
        err = fmctdc_get_time(tdc, &time_rb);
        if (err)
                return err;
```

Still about time, the user can add it's own offset without changing the timebase using *fmctdc_get_offset_user()* and *fmctdc_set_offset_user()*.

```
        err = fmctdc_set_offset_user(tdc, channel, offset_user);
        if (err)
                return err;
        err = fmctdc_get_offset_user(tdc, channel, &offset_user_rb);
        if (err)
                return err;
```

Finally, you can monitor the board temperature using *fmctdc_read_temperature()*, and pulse and timestamps statistics with *fmctdc_stats_recv_get()* and *fmctdc_stats_trans_get()*.

```
        err = fmctdc_stats_recv_get(tdc, channel, &recv);
        if (err)
                return err;
        err = fmctdc_stats_trans_get(tdc, channel, &trans);
        if (err)
                return err;
```

Note: If it can be useful there is one last status function in the API used to detect the transfer mode between the driver and the board. This function is *fmctdc_transfer_mode()*

Timestamp buffering has its own set of options. Buffering in hardware is fixed, it can't be configured, so what we are going to describe here is the Linux device driver buffering configuration. Because the TDC driver is based on ZIO, then you can choose the buffer allocator type. You can handle this option with the pair: *fmctdc_get_buffer_type()* and *fmctdc_set_buffer_type()*.

```
        err = fmctdc_set_buffer_type(tdc, buffer_type);
        if (err)
                return err;
        buffer_type_rb = fmctdc_get_buffer_type(tdc);
        if (buffer_type_rb < 0)
                return buffer_type_rb;
```

You can configure - and get - the buffer size (number of timestamps) with: *fmctdc_get_buffer_len()* and *fmctdc_set_buffer_len()*. Beware, that this function works only when using `FMCTDC_BUFFER_VMALLOC`.

```
        err = fmctdc_set_buffer_len(tdc, channel, buffer_len);
        if (err)
                return err;
        buffer_len_rb = fmctdc_get_buffer_len(tdc, channel);
        if (buffer_len_rb < 0)
                return buffer_len_rb;
```

Finally, you can select between to modes to handle buffer's overflows: `FMCTDC_BUFFER_CIRC` and `FMCTDC_BUFFER_FIFO`. The first will discard old timestamps to make space for the new ones, the latter will discard any new timestamp until the buffer get consumed. To configure this option you can use: *fmctdc_get_buffer_mode()* and *fmctdc_set_buffer_mode()*.

```
        err = fmctdc_set_buffer_mode(tdc, channel, buffer_mode);
        if (err)
                return err;
        buffer_mode_rb = fmctdc_get_buffer_mode(tdc, channel);
        if (buffer_mode_rb < 0)
                return buffer_mode_rb;
```

4.3.5 Acquisiton

Before actually being able to get timestamps, the TDC acquisition must be enabled. The acquisition can be *enabled* or *disabled* through its gateware using, respectivily, *fmctdc_channel_enable()* and *fmctdc_channel_disable()*.

```
        err = fmctdc_channel_enable(tdc, channel);
        if (err)
                return err;

        err = fetch_and_process(tdc);
        if (err)
                return err;

        err = fmctdc_channel_disable(tdc, channel);
        if (err)
                return err;
```

To read timestamps you may use functions *fmctdc_read()* and *fmctdc_fread()*. As the name may suggest, the first behaves like *read* and the second as *fread*.

```
        do {
                n = fmctdc_read(tdc, channel, ts, max, O_NONBLOCK);
        } while (n < 0 && errno == EAGAIN);
        if (n < 0)
                return n;
```

If you need to flush the buffer, you can use *fmctdc_flush()*.

```
        err = fmctdc_flush(tdc, channel);
        if (err)
                return err;
```

4.3.6 Timestamp Math

The TDC library API has functions to support timestamp math. They allow you to *add*, *subtract*, *normalize*, and *approximate*. These functions are: *fmctdc_ts_add()*, *fmctdc_ts_sub()*, *fmctdc_ts_norm()*, *fmctdc_ts_ps()*, and *fmctdc_ts_approx_ns()*.

4.4 The Library API

Defines

`PRItsps`
printf format for timestamps with pico-second resolution

`PRItspsVAL(_ts)`
printf value for timestamps with pico-second resolution

`PRItswr`
printf format for timestamps with White-Rabbit notation

`PRItswrVAL(_ts)`
printf value for timestamp with White-Rabbit notation

`__FMCTDC_ERR_MIN`

Enums

enum `fmctdc_error_numbers`
Values:

enumerator `FMCTDC_ERR_VMALLOC`

enumerator `FMCTDC_ERR_UNKNOWN_BUFFER_TYPE`

enumerator `FMCTDC_ERR_NOT_CONSISTENT_BUFFER_TYPE`

enumerator `FMCTDC_ERR_VERSION_MISMATCH`

enumerator `__FMCTDC_ERR_MAX`

enum `fmctdc_channel`
Enumeration for all TDC channels
Values:

enumerator `FMCTDC_CH_1`

enumerator `FMCTDC_CH_2`

enumerator `FMCTDC_CH_3`

enumerator `FMCTDC_CH_4`

enumerator **FMCTDC_CH_5**

enumerator **FMCTDC_CH_LAST**

enumerator **FMCTDC_NUM_CHANNELS**

enum **fmctdc_buffer_mode**

Enumeration of all buffer modes

Values:

enumerator **FMCTDC_BUFFER_FIFO**

FIFO policy: when buffer is full, new time-stamps will be dropped

enumerator **FMCTDC_BUFFER_CIRC**

circular buffer policy: when the buffer is full, old time-stamps will be overwritten by new ones

enum **fmctdc_buffer_type**

Enumeration of all buffer types

Values:

enumerator **FMCTDC_BUFFER_KMALLOC**

kernel allocator: kmalloc

enumerator **FMCTDC_BUFFER_VMALLOC**

kernel allocator: vmalloc

enum **fmctdc_channel_status**

Enumeration for all possible status of a channel

Values:

enumerator **FMCTDC_STATUS_DISABLE**

The cannel is disable

enumerator **FMCTDC_STATUS_ENABLE**

the channel is enable

enum **ft_transfer_mode**

Values:

enumerator **FT_ACQ_TYPE_FIFO**

enumerator **FT_ACQ_TYPE_DMA**

enum **fmctdc_ts_mode**

Enumeration for all possible time-stmap mode

Values:

enumerator **FMCTDC_TS_MODE_POST**

after post-processing

enumerator **FMCTDC_TS_MODE_RAW**

directly from ACAM chip. This should be used ONLY when debugging low level issues

Functions

const char ***fmctdc_strerror**(int err)

It returns the error message associated to the given error code

Parameters

err – **[in]** error code

int **fmctdc_init**(void)

Init the library. You must call this function before use any other library function.

Returns

0 on success, otherwise -1 and errno is appropriately set

void **fmctdc_exit**(void)

It releases all the resources used by the library and allocated by fmctdc_init().

int **fmctdc_set_time**(struct fmctdc_board *b, const struct *fmctdc_time* *t)

It sets the TDC base-time according to the given time-stamp. Note that, for the time being, it sets only seconds. Note that, you can set the time only when the acquisition is disabled.

Parameters

- **userb** – **[in]** TDC board instance token
- **t** – **[in]** time-stamp

Returns

0 on success, otherwise -1 and errno is set

int **fmctdc_get_time**(struct fmctdc_board *b, struct *fmctdc_time* *t)

It gets the base-time of a TDC device. Note that, for the time being, it gets only seconds.

Parameters

- **userb** – **[in]** TDC board instance token
- **t** – **[out]** time-stamp

Returns

0 on success, otherwise -1 and errno is set

int **fmctdc_set_host_time**(struct fmctdc_board *b)

It sets the TDC base-time according to the host time

Parameters

userb – **[in]** TDC board instance token

Returns
0 on success, otherwise -1 and errno is set appropriately

int `fmctdc_wr_mode`(struct fmctdc_board *b, int on)

It enables/disables the WhiteRabbit timing system on a TDC device

Parameters

- `userb` – **[in]** TDC board instance token
- `on` – **[in]** white-rabbit status to set

Returns
0 on success, otherwise -1 and errno is set appropriately

int `fmctdc_check_wr_mode`(struct fmctdc_board *b)

It check the current status of the WhiteRabbit timing system on a TDC device

Parameters
`userb` – **[in]** TDC board instance token

Returns
0 if it properly works, -1 on error and errno is set appropriately.

- ENOLINK if it is not synchronized and
- ENODEV if it is not enabled

float `fmctdc_read_temperature`(struct fmctdc_board *b)

It reads the current temperature of a TDC device

Parameters
`userb` – **[in]** TDC board instance token

Returns
temperature

int `fmctdc_channel_status_set`(struct fmctdc_board *userb, unsigned int channel, enum *fmctdc_channel_status* status)

The function enables/disables timestamp acquisition for the given channel.

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** channel to which we want change status
- `status` – **[in]** enable status to set

Returns
0 on success, otherwise -1 and errno is set appropriately

int `fmctdc_channel_enable`(struct fmctdc_board *userb, unsigned int channel)

It enables a given channel. NOTE: it is just a wrapper of fmctdc_channel_status_set()

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** channel to which we want change status

Returns
0 on success, otherwise -1 and errno is set appropriately

int **`fmctdc_channel_disable`**(struct fmctdc_board *userb, unsigned int channel)

It disable a given channel. NOTE: it is just a wrapper of fmctdc_channel_status_set()

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** channel to which we want change status

Returns

0 on success, otherwise -1 and errno is set appropriately

int **`fmctdc_channel_status_get`**(struct fmctdc_board *userb, unsigned int channel)

It gets the acquisition status of a TDC channel

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** channel to which we want read the status

Returns

the acquisition status (0 disabled, 1 enabled), otherwise -1 and errno is set appropriately

int **`fmctdc_set_termination`**(struct fmctdc_board *b, unsigned int channel, int enable)

The function enables/disables the 50 Ohm termination of the given channel. Termination may be changed anytime.

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** to use
- **`on`** – **[in]** status of the termination to set

Returns

0 on success, otherwise a negative errno code is set appropriately

int **`fmctdc_get_termination`**(struct fmctdc_board *b, unsigned int channel)

The function returns current temrmination status: 0 if the given channel is high-impedance and positive if it is 50 Ohm-terminated.

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** to use

Returns

termination status, otherwise a negative errno code is set appropriately

int **`fmctdc_get_buffer_type`**(struct fmctdc_board *userb)

The function returns current buffer type: 0 for kmallo, 1 for vmalloc.

Parameters

`userb` – **[in]** TDC board instance token

Returns

buffer type, otherwise a negative errno code is set appropriately

int **`fmctdc_set_buffer_type`**(struct fmctdc_board *userb, enum *fmctdc_buffer_type* type)

The function sets the buffer type for a device

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`type`** – **[in]** buffer type to use

Returns
0 on success, otherwise a negative errno code is set appropriately

int **`fmctdc_get_buffer_mode`**(struct fmctdc_board *userb, unsigned int channel)

The function returns current buffer mode: 0 for FIFO, 1 for circular buffer.

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** to use

Returns
buffer mode, otherwise a negative errno code is set appropriately

int **`fmctdc_set_buffer_mode`**(struct fmctdc_board *userb, unsigned int channel, enum *fmctdc_buffer_mode* mode)

The function sets the buffer mode for a channel

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** to use
- **`mode`** – **[in]** buffer mode to use

Returns
0 on success, otherwise a negative errno code is set appropriately

int **`fmctdc_get_buffer_len`**(struct fmctdc_board *userb, unsigned int channel)

The function returns current driver buffer length (number of timestamps)

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** to use

Returns
buffer lenght, otherwise a negative errno code is set appropriately

int **`fmctdc_set_buffer_len`**(struct fmctdc_board *userb, unsigned int channel, unsigned int length)

The function set the buffer length

Internally, the buffer allocates memory in chunks of minimun 1KiB. This means, for example, that if you ask for 65 timestamp the buffer will allocate space for 128. This because 64 timestamps fit in 1KiB, to store 65 we need 2KiB (128 timestamps).

NOTE: it works only with the VMALLOC allocator.

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** to use
- **`length`** – **[in]** maximum number of timestamps to store (min: 64)

Returns
0 on success, otherwise a negative errno code is set appropriately

int `fmctdc_set_offset_user`(struct fmctdc_board *userb, unsigned int channel, int32_t offset)

It sets the user offset to be applied on incoming timestamps. All the timestamps read from the driver (this means also from this library) will be already corrected using this offset.

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** target channel [0, 4]
- `offset` – **[in]** the number of pico-seconds to be added

Returns

0 on success, otherwise -1 and errno is set appropriately

int `fmctdc_get_offset_user`(struct fmctdc_board *userb, unsigned int channel, int32_t *offset)

It get the current user offset applied to the incoming timestamps

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** target channel [0, 4]
- `offset` – **[out]** the number of pico-seconds to be added

Returns

0 on success, otherwise -1 and errno is set appropriately

int `fmctdc_transfer_mode`(struct fmctdc_board *userb, enum *ft_transfer_mode* *mode)

It gets the current transfer mode

Parameters

- `userb` – **[in]** TDC board instance token
- `mode` – **[out]** transfer mode

Returns

0 on success, otherwise -1 and errno is set appropriately

int `fmctdc_coalescing_timeout_set`(struct fmctdc_board *userb, unsigned int channel, unsigned int timeout_ms)

It sets the coalescing timeout on a given channel

It does not work per-channel for the following acquisition mechanism:

- FIFO (it will return the global IRQ coalescing timeout)

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** target channel [0, 4]
- `timeout_ms` – **[in]** ms timeout to trigger IRQ

Returns

0 on success, otherwise -1 and errno is set appropriately

int **`fmctdc_coalescing_timeout_get`**(struct fmctdc_board *userb, unsigned int channel, unsigned int *timeout_ms)

It gets the coalescing timeout from a given channel

It does not work per-channel for the following acuqisition mechanism:

- FIFO: there is a global configuration for all channels

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** target channel [0, 4]
- **`timeout_ms`** – **[out]** ms timeout to trigger IRQ

Returns

0 on success, otherwise -1 and errno is set appropriately

int **`fmctdc_ts_mode_set`**(struct fmctdc_board *userb, unsigned int channel, enum *fmctdc_ts_mode* mode)

It sets the timestamp mode

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** target channel [0, 4]
- **`mode`** – **[in]** time-stamp mode

Returns

0 on success, otherwise -1 and errno is set appropriately

int **`fmctdc_ts_mode_get`**(struct fmctdc_board *userb, unsigned int channel, enum *fmctdc_ts_mode* *mode)

It gets the timestamp mode

Parameters

- **`userb`** – **[in]** TDC board instance token
- **`channel`** – **[in]** target channel [0, 4]
- **`mode`** – **[out]** time-stamp mode

Returns

0 on success, otherwise -1 and errno is set appropriately

struct fmctdc_board ***`fmctdc_open`**(int dev_id)

struct fmctdc_board ***`fmctdc_open_by_lun`**(int lun)

It opens one specific device by logical unit number (CERN/BE-CO-like). The function uses a symbolic link in /dev that points to the standard device. The link is created by the local installation procedure, and it allows to get the device id according to the LUN. Read also fmctdc_open() documentation.

Parameters

`lun` – **[in]** Logical Unit Number

Returns

an instance token, otherwise NULL and errno is appripriately set int `fmctdc_close`(struct fmctdc_board*)

It closes a TDC instance opened with fmctdc_open() or fmctdc_open_by_lun()

Parameters
`userb` – **[in]** TDC board instance token

Returns
0 on success, otherwise -1 and errno is set appropriately

int `fmctdc_fread`(struct fmctdc_board *b, unsigned int channel, struct *fmctdc_time* *t, int n)

this "fread" behaves like stdio: it reads all the samples. Read fmctdc_read() for more details about the function.

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** channel to use
- `t` – **[out]** array of time-stamps
- `n` – **[in]** number of elements to save in the array

Returns
number of acquired time-stamps, otherwise -1 and errno is set appropriately

int `fmctdc_fileno_channel`(struct fmctdc_board *b, unsigned int channel)

It get the file descriptor of a TDC channel. So, for example, you can poll(2) and select(2). Note that, the file descriptor is the file-descriptor of a ZIO control char-device.

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** channel to use

Returns
a file descriptor, otherwise -1 and errno is set appropriately

int `fmctdc_read`(struct fmctdc_board *b, unsigned int channel, struct *fmctdc_time* *t, int n, int flags)

It reads a given number of time-stamps from the driver. It will wait at most once and return the number of samples that it received from a given input channel.

Timestamps are to the base time.

This "read" behaves like the system call and obeys O_NONBLOCK

Parameters

- `userb` – **[in]** TDC board instance token
- `channel` – **[in]** channel to use [0, 4]
- `t` – **[out]** array of time-stamps
- `n` – **[in]** number of elements to save in the array
- `flags` – **[in]** tune the behaviour of the function. O_NONBLOCK - do not block

Returns
number of acquired time-stamps, otherwise -1 and errno is set appropriately.

- EINVAL for invalid arguments
- EIO for invalid IO transfer
- EAGAIN if nothing ready to read in NONBLOCK mode

int **fmctdc_flush**(struct fmctdc_board *userb, unsigned int channel)

It removes all samples from the channel buffer. In order to doing this, the function temporary disable any active acquisition, only when the flush is completed the acquisition will be re-enabled

Parameters

- **userb** – **[in]** TDC board instance token
- **channel** – **[in]** target channel [0, 4]

Returns
0 on success, otherwise -1 and errno is set appropriately

int **fmctdc_stats_recv_get**(struct fmctdc_board *userb, unsigned int channel, uint32_t *val)

It gets the number of received pulses (on hardware)

Parameters

- **userb** – **[in]** TDC board instance token
- **channel** – **[in]** target channel [0, 4]
- **val** – **[out]** number of received pulses

Returns
0 on success, otherwise -1 and errno is set appropriately

int **fmctdc_stats_trans_get**(struct fmctdc_board *userb, unsigned int channel, uint32_t *val)

It gets the number of transferred timestamps

Parameters

- **userb** – **[in]** TDC board instance token
- **channel** – **[in]** target channel [0, 4]
- **val** – **[out]** number of transferred timestamps

Returns
0 on success, otherwise -1 and errno is set appropriately

uint64_t **fmctdc_ts_approx_ns**(struct *fmctdc_time* *a)

Set of mathematical functions on time-stamps

It provides a nano-second approximation of the timestamp.

Parameters
a – **[in]** timestamp

Returns
it returns the time stamp in nano-seconds

uint64_t **fmctdc_ts_ps**(struct *fmctdc_time* *a)

It provides a pico-seconds representation of the time stamp. Bear in mind that it may overflow. If you thing that it may happen, check the timestamp

Parameters
a – **[in]** timestamp

Returns
it returns the time stamp in pico-seconds

void **fmctdc_ts_norm**(struct *fmctdc_time* *a)

It normalizes the timestamp

Parameters

a – **[inout]** timestamp

int **fmctdc_ts_sub**(struct *fmctdc_time* *r, const struct *fmctdc_time* *a, const struct *fmctdc_time* *b)

It perform the subtraction: r = a - b

Parameters

- **r** – **[out]** result
- **a** – **[in]** normalized timestamp
- **b** – **[in]** normalized timestamp

Returns

1 if the difference is negative, otherwise 0

void **fmctdc_ts_add**(struct *fmctdc_time* *r, const struct *fmctdc_time* *a, const struct *fmctdc_time* *b)

It perform an addiction: r = a + b

Parameters

- **r** – **[out]** result
- **a** – **[in]** normalized timestamp
- **b** – **[in]** normalized timestamp

int **_fmctdc_tscmp**(struct *fmctdc_time* *a, struct *fmctdc_time* *b)

Variables

const char *const **libfmctdc_version_s**

libfmctdc version string

const char *const **libfmctdc_zio_version_s**

zio version string used during compilation of libfmctdc

struct **fmctdc_time**

#include <fmctdc-lib.h> FMC-TDC time-stamp descriptor

Public Members

uint64_t **seconds**

TAI seconds. Note this is *not* an UTC time; the counter does not support leap seconds. The internal counter is also limited to 32 bits (2038-error-prone).

uint32_t **coarse**

number of ticks of 8ns since the beginning of the last second uint32_t **frac**

fractional part of an 8 ns tick, rescaled to (0..4095) range - i.e. 0 = 0 ns, and 4095 = 7.999 ns.

uint32_t **seq_id**

channel sequence number

uint32_t **debug**

debug stuff, driver/firmware-specific

CHAPTER

FIVE

THE MEMORY MAP

5.1 Supported Designs

Here you can find the complete memory MAP for the supported designs. This will include the TDC registers as well as the carrier registers and any other component used in an FMC-TDC-1NS-5CH design.

5.1.1 SPEC FMC-TDC-1NS-5CHA

The memory map is divided in two parts: the *Carrier (SPEC)* part common to all SPEC designs, and the *TDC* part specific to the FMC-TDC-1NS-5CHA mezzanine.

Memory map summary

SPEC FMC-TDC-1NS-5CHA memory map

HW address	Type	Name	HDL name
0x00000-0x01fff	SUBMAP	spec-base-regs	spec-base-regs
0x10000-0x1ffff	SUBMAP	tdc-base-regs	tdc-base-regs

Registers description

SPEC base registers

Memory map summary

SPEC base registers

HW address	Type	Name	HDL name
0x0000-0x003f	SUBMAP	metadata	metadata
0x0040-0x005f	BLOCK	csr	csr
0x0040	REG	csr.app_offset	csr_app_offset
0x0044	REG	csr.resets	csr_resets
0x0048	REG	csr.fmc_presence	csr_fmc_presence
0x004c	REG	csr.gn4124_status	csr_gn4124_status
0x0050	REG	csr.ddr_status	csr_ddr_status
0x0054	REG	csr.pcb_rev	csr_pcb_rev
0x0070-0x007f	SUBMAP	therm_id	therm_id
0x0080-0x009f	SUBMAP	fmc_i2c	fmc_i2c
0x00a0-0x00bf	SUBMAP	flash_spi	flash_spi
0x00c0-0x00ff	SUBMAP	dma	dma
0x0100-0x01ff	SUBMAP	vic	vic
0x0200-0x02ff	SUBMAP	buildinfo	buildinfo
0x1000-0x1fff	SUBMAP	wrc_regs	wrc_regs

Registers description

csr.app_offset

- HDL name: csr_app_offset
- address: 0x40
- block offset: 0x0
- access mode: ro

offset to the application metadata

31	30	29	28	27	26	25	24
app_offset[31:24]							
23	22	21	20	19	18	17	16
app_offset[23:16]							
15	14	13	12	11	10	9	8
app_offset[15:8]							
7	6	5	4	3	2	1	0
app_offset[7:0]							

csr.resets

- HDL name: csr_resets
- address: 0x44
- block offset: 0x4
- access mode: rw

global and application resets

31	30	29	28	27	26	25	24
•	•	•	•	•	•	•	•
23	22	21	20	19	18	17	16
•	•	•	•	•	•	•	•
15	14	13	12	11	10	9	8
•	•	•	•	•	•	•	•
7	6	5	4	3	2	1	0
•	•	•	•	•	•	appl	global

global
(not documented)

appl
(not documented)

csr.fmc_presence

- HDL name: csr_fmc_presence
- address: 0x48
- block offset: 0x8
- access mode: ro

presence lines for the fmcs

31	30	29	28	27	26	25	24
fmc_presence[31:24]							
23	22	21	20	19	18	17	16
fmc_presence[23:16]							
15	14	13	12	11	10	9	8
fmc_presence[15:8]							
7	6	5	4	3	2	1	0
fmc_presence[7:0]							

csr.gn4124_status

- HDL name: csr_gn4124_status
- address: 0x4c
- block offset: 0xc
- access mode: ro

status of gennum

31	30	29	28	27	26	25	24
gn4124_status[31:24]							
23	22	21	20	19	18	17	16
gn4124_status[23:16]							
15	14	13	12	11	10	9	8
gn4124_status[15:8]							
7	6	5	4	3	2	1	0
gn4124_status[7:0]							

csr.ddr_status

- HDL name: csr_ddr_status
- address: 0x50
- block offset: 0x10
- access mode: ro

status of the ddr3 controller

31	30	29	28	27	26	25	24
•	•	•	•	•	•	•	•
23	22	21	20	19	18	17	16
•	•	•	•	•	•	•	•
15	14	13	12	11	10	9	8
•	•	•	•	•	•	•	•
7	6	5	4	3	2	1	0
•	•	•	•	•	•	•	calib_done

calib_done
Set when calibration is done.

csr.pcb_rev

- HDL name: csr_pcb_rev
- address: 0x54
- block offset: 0x14
- access mode: ro

pcb revision

31	30	29	28	27	26	25	24
•	•	•	•	•	•	•	•
23	22	21	20	19	18	17	16
•	•	•	•	•	•	•	•
15	14	13	12	11	10	9	8
•	•	•	•	•	•	•	•
7	6	5	4	3	2	1	0
•	•	•	•	rev[3:0]			

rev

(not documented)

FMC-TDC-1NS-5CHA

See *TDC memory map*.

5.1.2 SVEC FMC-TDC-1NS-5CHA

The memory map is divided in two parts: the *Carrier (SVEC)* part common to all SVEC designs, and two memory regions for TDCs (*TDC1* and *TDC2*) part specific to the FMC-TDC-1NS-5CHA mezzanine.

Memory map summary

SVEC FMC-TDC-1NS-5CHA memory map

HW address	Type	Name	HDL name
0x00000-0x0ffff	SUBMAP	svec-base-regs	svec-base-regs
0x10000-0x1ffff	SUBMAP	tdc1-base-regs	tdc1-base-regs
0x20000-0x2ffff	SUBMAP	tdc2-base-regs	tdc2-base-regs

Registers description

SVEC base registers

Memory map summary

SVEC base registers

HW address	Type	Name	HDL name
0x0000-0x003f	SUBMAP	metadata	metadata
0x0040-0x005f	BLOCK	csr	csr
0x0040	REG	csr.app_offset	csr_app_offset
0x0044	REG	csr.resets	csr_resets
0x0048	REG	csr.fmc_presence	csr_fmc_presence
0x004c	REG	csr.unused0	csr_unused0
0x0050	REG	csr.ddr_status	csr_ddr_status
0x0054	REG	csr.pcb_rev	csr_pcb_rev
0x0058	REG	csr.ddr4_addr	csr_ddr4_addr
0x005c	REG	csr.ddr5_addr	csr_ddr5_addr
0x0080-0x008f	SUBMAP	therm_id	therm_id
0x00a0-0x00bf	SUBMAP	fmc_i2c	fmc_i2c
0x00c0-0x00df	SUBMAP	flash_spi	flash_spi
0x0100-0x01ff	SUBMAP	vic	vic
0x0200-0x02ff	SUBMAP	buildinfo	buildinfo
0x1000-0x17ff	SUBMAP	wrc_regs	wrc_regs
0x2000-0x2fff	SUBMAP	ddr4_data	ddr4_data
0x3000-0x3fff	SUBMAP	ddr5_data	ddr5_data

Registers description

csr.app_offset

- HDL name: csr_app_offset
- address: 0x40
- block offset: 0x0
- access mode: ro

offset to the application metadata

31	30	29	28	27	26	25	24
app_offset[31:24]							
23	22	21	20	19	18	17	16
app_offset[23:16]							
15	14	13	12	11	10	9	8
app_offset[15:8]							
7	6	5	4	3	2	1	0
app_offset[7:0]							

csr.resets

- HDL name: csr_resets
- address: 0x44
- block offset: 0x4
- access mode: rw

global and application resets

31	30	29	28	27	26	25	24
•	•	•	•	•	•	•	•
23	22	21	20	19	18	17	16
•	•	•	•	•	•	•	•
15	14	13	12	11	10	9	8
•	•	•	•	•	•	•	•
7	6	5	4	3	2	1	0
•	•	•	•	•	•	appl	global

global
(not documented)

appl
(not documented)

csr.fmc_presence

- HDL name: csr_fmc_presence
- address: 0x48
- block offset: 0x8
- access mode: ro

presence lines for the fmcs

31	30	29	28	27	26	25	24
fmc_presence[31:24]							
23	22	21	20	19	18	17	16
fmc_presence[23:16]							
15	14	13	12	11	10	9	8
fmc_presence[15:8]							
7	6	5	4	3	2	1	0
fmc_presence[7:0]							

csr.unused0

- HDL name: csr_unused0
- address: 0x4c
- block offset: 0xc
- access mode: ro

unused (status of gennum)

31	30	29	28	27	26	25	24
unused0[31:24]							
23	22	21	20	19	18	17	16
unused0[23:16]							
15	14	13	12	11	10	9	8
unused0[15:8]							
7	6	5	4	3	2	1	0
unused0[7:0]							

csr.ddr_status

- HDL name: csr_ddr_status
- address: 0x50
- block offset: 0x10
- access mode: ro

status of the ddr controllers

31	30	29	28	27	26	25	24
•	•	•	•	•	•	•	•
23	22	21	20	19	18	17	16
•	•	•	•	•	•	•	•
15	14	13	12	11	10	9	8
•	•	•	•	•	•	•	•
7	6	5	4	3	2	1	0
•	•	•	•	•	•	ddr5_calib_done	ddr4_calib_done

ddr4_calib_done
Set when ddr4 calibration is done.

ddr5_calib_done
Set when ddr5 calibration is done.

csr.pcb_rev

- HDL name: csr_pcb_rev
- address: 0x54
- block offset: 0x14
- access mode: ro

pcb revision

31	30	29	28	27	26	25	24
•	•	•	•	•	•	•	•
23	22	21	20	19	18	17	16
•	•	•	•	•	•	•	•
15	14	13	12	11	10	9	8
•	•	•	•	•	•	•	•
7	6	5	4	3	2	1	0
•	•	•	rev[4:0]				

rev
(not documented)

csr.ddr4_addr

- HDL name: csr_ddr4_addr
- address: 0x58
- block offset: 0x18
- access mode: rw

address of data to read or to write

31	30	29	28	27	26	25	24
ddr4_addr[31:24]							
23	22	21	20	19	18	17	16
ddr4_addr[23:16]							
15	14	13	12	11	10	9	8
ddr4_addr[15:8]							
7	6	5	4	3	2	1	0
ddr4_addr[7:0]							

csr.ddr5_addr

- HDL name: csr_ddr5_addr
- address: 0x5c
- block offset: 0x1c
- access mode: rw

address of data to read or to write

31	30	29	28	27	26	25	24
ddr5_addr[31:24]							
23	22	21	20	19	18	17	16
ddr5_addr[23:16]							
15	14	13	12	11	10	9	8
ddr5_addr[15:8]							
7	6	5	4	3	2	1	0
ddr5_addr[7:0]							

First FMC-TDC-1NS-5CHA

See *TDC memory map*.

Second FMC-TDC-1NS-5CHA

See *TDC memory map*.

5.2 TDC memory map

Following the memory map for the part of the TDC design that drives the FMC-TDC-1NS-5CH modules.

5.2.1 Memory map summary

FMC-TDC-1NS-5CH mezzanine memory map

HW address	Type	Name	HDL name
0x1000-0x1fff	SUBMAP	one-wire	one-wire
0x2000-0x2fff	SUBMAP	core	core
0x3000-0x3fff	SUBMAP	eic	eic
0x4000-0x4fff	SUBMAP	i2c	i2c
0x5000-0x5fff	SUBMAP	mem	mem
0x6000-0x6fff	SUBMAP	mem-dma	mem-dma
0x7000-0x7fff	SUBMAP	mem-dma-eic	mem-dma-eic

Registers description

5.2.2 One wire

5.2.3 TDC Onewire Master

Memory map summary

SW Offset	Type	Name	HW prefix	C prefix
0x0	REG	Status Register	tdc_ow_csr	CSR
0x4	REG	Board Temperature	tdc_ow_temp	TEMP
0x8	REG	Board Unique ID (MSW)	tdc_ow_id_h	ID_H
0xc	REG	Board Unique ID (LSW)	tdc_ow_id_l	ID_L

Register description

Status Register

HW prefix:	tdc_ow_csr
HW address:	0x0
SW prefix:	CSR
SW offset:	0x0

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	VALID

- **VALID** [*read/write*]: Temperature and ID valid
  read 1: the values in the TEMP, ID_H, ID_L registers contain a valid readout from the DS18xx chip

Board Temperature

HW prefix:	tdc_ow_temp
HW address:	0x1
SW prefix:	TEMP
SW offset:	0x4

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
TEMP[15:8]							
7	6	5	4	3	2	1	0
TEMP[7:0]							

- **TEMP** [*read-only*]: Temperature

Board Unique ID (MSW)

HW prefix:	tdc_ow_id_h
HW address:	0x2
SW prefix:	ID_H
SW offset:	0x8

31	30	29	28	27	26	25	24
ID_H[31:24]							
23	22	21	20	19	18	17	16
ID_H[23:16]							
15	14	13	12	11	10	9	8
ID_H[15:8]							
7	6	5	4	3	2	1	0
ID_H[7:0]							

- **ID_H** [*read-only*]: Unique ID (32 highest bits)

Board Unique ID (LSW)

HW prefix:	tdc_ow_id_l
HW address:	0x3
SW prefix:	ID_L
SW offset:	0xc

31	30	29	28	27	26	25	24
ID_L[31:24]							
23	22	21	20	19	18	17	16
ID_L[23:16]							
15	14	13	12	11	10	9	8
ID_L[15:8]							
7	6	5	4	3	2	1	0
ID_L[7:0]							

- **ID_L** [*read-only*]: Unique ID (32 lowest bits)

5.2.4 Core

5.2.5 EIC

5.2.6 TDC EIC

FMC TDC embedded interrrupt controller.

Memory map summary

SW Offset	Type	Name	HW prefix	C prefix
0x20	REG	Interrupt disable register	tdc_eic_eic_idr	EIC_IDR
0x24	REG	Interrupt enable register	tdc_eic_eic_ier	EIC_IER
0x28	REG	Interrupt mask register	tdc_eic_eic_imr	EIC_IMR
0x2c	REG	Interrupt status register	tdc_eic_eic_isr	EIC_ISR

Register description

Interrupt disable register

HW prefix:	tdc_eic_eic_idr
HW address:	0x8
SW prefix:	EIC_IDR
SW offset:	0x20

Writing 1 disables handling of the interrupt associated with corresponding bit. Writin 0 has no effect.

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	TDC_DMA5	TDC_DMA4
7	6	5	4	3	2	1	0
TDC_DMA3	TDC_DMA2	TDC_DMA1	TDC_FIFO5	TDC_FIFO4	TDC_FIFO3	TDC_FIFO2	TDC_FIFO1

- **TDC_FIFO1** [*write-only*]: FMC TDC timestamps interrupt (FIFO1)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (FIFO1)'
  write 0: no effect
- **TDC_FIFO2** [*write-only*]: FMC TDC timestamps interrupt (FIFO2)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (FIFO2)'
  write 0: no effect

- **TDC_FIFO3** [*write-only*]: FMC TDC timestamps interrupt (FIFO3)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (FIFO3)'
  write 0: no effect
- **TDC_FIFO4** [*write-only*]: FMC TDC timestamps interrupt (FIFO4)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (FIFO4)'
  write 0: no effect
- **TDC_FIFO5** [*write-only*]: FMC TDC timestamps interrupt (FIFO5)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (FIFO5)'
  write 0: no effect
- **TDC_DMA1** [*write-only*]: FMC TDC timestamps interrupt (DMA1)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (DMA1)'
  write 0: no effect
- **TDC_DMA2** [*write-only*]: FMC TDC timestamps interrupt (DMA2)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (DMA2)'
  write 0: no effect
- **TDC_DMA3** [*write-only*]: FMC TDC timestamps interrupt (DMA3)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (DMA3)'
  write 0: no effect
- **TDC_DMA4** [*write-only*]: FMC TDC timestamps interrupt (DMA4)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (DMA4)'
  write 0: no effect
- **TDC_DMA5** [*write-only*]: FMC TDC timestamps interrupt (DMA5)
  write 1: disable interrupt 'FMC TDC timestamps interrupt (DMA5)'
  write 0: no effect

Interrupt enable register

HW prefix:	tdc_eic_eic_ier
HW address:	0x9
SW prefix:	EIC_IER
SW offset:	0x24

Writing 1 enables handling of the interrupt associated with corresponding bit. Writin 0 has no effect.

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	TDC_DMA5	TDC_DMA4
7	6	5	4	3	2	1	0
TDC_DMA3	TDC_DMA2	TDC_DMA1	TDC_FIFO5	TDC_FIFO4	TDC_FIFO3	TDC_FIFO2	TDC_FIFO1

- **TDC_FIFO1** [*write-only*]: FMC TDC timestamps interrupt (FIFO1)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (FIFO1)'
  write 0: no effect

- **TDC_FIFO2** [*write-only*]: FMC TDC timestamps interrupt (FIFO2)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (FIFO2)'
  write 0: no effect
- **TDC_FIFO3** [*write-only*]: FMC TDC timestamps interrupt (FIFO3)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (FIFO3)'
  write 0: no effect
- **TDC_FIFO4** [*write-only*]: FMC TDC timestamps interrupt (FIFO4)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (FIFO4)'
  write 0: no effect
- **TDC_FIFO5** [*write-only*]: FMC TDC timestamps interrupt (FIFO5)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (FIFO5)'
  write 0: no effect
- **TDC_DMA1** [*write-only*]: FMC TDC timestamps interrupt (DMA1)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (DMA1)'
  write 0: no effect
- **TDC_DMA2** [*write-only*]: FMC TDC timestamps interrupt (DMA2)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (DMA2)'
  write 0: no effect
- **TDC_DMA3** [*write-only*]: FMC TDC timestamps interrupt (DMA3)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (DMA3)'
  write 0: no effect
- **TDC_DMA4** [*write-only*]: FMC TDC timestamps interrupt (DMA4)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (DMA4)'
  write 0: no effect
- **TDC_DMA5** [*write-only*]: FMC TDC timestamps interrupt (DMA5)
  write 1: enable interrupt 'FMC TDC timestamps interrupt (DMA5)'
  write 0: no effect

Interrupt mask register

HW prefix:	tdc_eic_eic_imr
HW address:	0xa
SW prefix:	EIC_IMR
SW offset:	0x28

Shows which interrupts are enabled. 1 means that the interrupt associated with the bitfield is enabled

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	TDC_DMA5	TDC_DMA4
7	6	5	4	3	2	1	0
TDC_DMA3	TDC_DMA2	TDC_DMA1	TDC_FIFO5	TDC_FIFO4	TDC_FIFO3	TDC_FIFO2	TDC_FIFO1

- **TDC_FIFO1** [*read-only*]: FMC TDC timestamps interrupt (FIFO1)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO1)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (FIFO1)' is disabled
- **TDC_FIFO2** [*read-only*]: FMC TDC timestamps interrupt (FIFO2)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO2)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (FIFO2)' is disabled
- **TDC_FIFO3** [*read-only*]: FMC TDC timestamps interrupt (FIFO3)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO3)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (FIFO3)' is disabled
- **TDC_FIFO4** [*read-only*]: FMC TDC timestamps interrupt (FIFO4)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO4)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (FIFO4)' is disabled
- **TDC_FIFO5** [*read-only*]: FMC TDC timestamps interrupt (FIFO5)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO5)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (FIFO5)' is disabled
- **TDC_DMA1** [*read-only*]: FMC TDC timestamps interrupt (DMA1)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA1)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (DMA1)' is disabled
- **TDC_DMA2** [*read-only*]: FMC TDC timestamps interrupt (DMA2)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA2)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (DMA2)' is disabled
- **TDC_DMA3** [*read-only*]: FMC TDC timestamps interrupt (DMA3)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA3)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (DMA3)' is disabled
- **TDC_DMA4** [*read-only*]: FMC TDC timestamps interrupt (DMA4)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA4)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (DMA4)' is disabled
- **TDC_DMA5** [*read-only*]: FMC TDC timestamps interrupt (DMA5)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA5)' is enabled
  read 0: interrupt 'FMC TDC timestamps interrupt (DMA5)' is disabled

Interrupt status register

HW prefix:	tdc_eic_eic_isr
HW address:	0xb
SW prefix:	EIC_ISR
SW offset:	0x2c

Each bit represents the state of corresponding interrupt. 1 means the interrupt is pending. Writing 1 to a bit clears the corresponding interrupt. Writing 0 has no effect.

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	TDC_DMA5	TDC_DMA4
7	6	5	4	3	2	1	0
TDC_DMA3	TDC_DMA2	TDC_DMA1	TDC_FIFO5	TDC_FIFO4	TDC_FIFO3	TDC_FIFO2	TDC_FIFO1

- **TDC_FIFO1** [*read/write*]: FMC TDC timestamps interrupt (FIFO1)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO1)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (FIFO1)'
  write 0: no effect
- **TDC_FIFO2** [*read/write*]: FMC TDC timestamps interrupt (FIFO2)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO2)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (FIFO2)'
  write 0: no effect
- **TDC_FIFO3** [*read/write*]: FMC TDC timestamps interrupt (FIFO3)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO3)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (FIFO3)'
  write 0: no effect
- **TDC_FIFO4** [*read/write*]: FMC TDC timestamps interrupt (FIFO4)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO4)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (FIFO4)'
  write 0: no effect
- **TDC_FIFO5** [*read/write*]: FMC TDC timestamps interrupt (FIFO5)
  read 1: interrupt 'FMC TDC timestamps interrupt (FIFO5)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (FIFO5)'
  write 0: no effect
- **TDC_DMA1** [*read/write*]: FMC TDC timestamps interrupt (DMA1)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA1)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (DMA1)'
  write 0: no effect
- **TDC_DMA2** [*read/write*]: FMC TDC timestamps interrupt (DMA2)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA2)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (DMA2)'
  write 0: no effect
- **TDC_DMA3** [*read/write*]: FMC TDC timestamps interrupt (DMA3)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA3)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (DMA3)'
  write 0: no effect
- **TDC_DMA4** [*read/write*]: FMC TDC timestamps interrupt (DMA4)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA4)' is pending

read 0: interrupt not pending
write 1: clear interrupt 'FMC TDC timestamps interrupt (DMA4)'
write 0: no effect

- **TDC_DMA5** [*read/write*]: FMC TDC timestamps interrupt (DMA5)
  read 1: interrupt 'FMC TDC timestamps interrupt (DMA5)' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'FMC TDC timestamps interrupt (DMA5)'
  write 0: no effect

Interrupts

FMC TDC timestamps interrupt (FIFO1)

HW prefix:	tdc_eic_tdc_fifo1
C prefix:	TDC_FIFO1
Trigger:	high level

FMC TDC FIFO1 not empty.

FMC TDC timestamps interrupt (FIFO2)

HW prefix:	tdc_eic_tdc_fifo2
C prefix:	TDC_FIFO2
Trigger:	high level

FMC TDC FIFO1 not empty.

FMC TDC timestamps interrupt (FIFO3)

HW prefix:	tdc_eic_tdc_fifo3
C prefix:	TDC_FIFO3
Trigger:	high level

FMC TDC FIFO3 not empty.

FMC TDC timestamps interrupt (FIFO4)

HW prefix:	tdc_eic_tdc_fifo4
C prefix:	TDC_FIFO4
Trigger:	high level

FMC TDC FIFO4 not empty.

FMC TDC timestamps interrupt (FIFO5)

HW prefix:	tdc_eic_tdc_fifo5
C prefix:	TDC_FIFO5
Trigger:	high level

FMC TDC FIFO5 not empty.

FMC TDC timestamps interrupt (DMA1)

HW prefix:	tdc_eic_tdc_dma1
C prefix:	TDC_DMA1
Trigger:	high level

FMC TDC DMA1 acquisition ready.

FMC TDC timestamps interrupt (DMA2)

HW prefix:	tdc_eic_tdc_dma2
C prefix:	TDC_DMA2
Trigger:	high level

FMC TDC DMA1 acquisition ready.

FMC TDC timestamps interrupt (DMA3)

HW prefix:	tdc_eic_tdc_dma3
C prefix:	TDC_DMA3
Trigger:	high level

FMC TDC DMA3 acquisition ready.

FMC TDC timestamps interrupt (DMA4)

HW prefix:	tdc_eic_tdc_dma4
C prefix:	TDC_DMA4
Trigger:	high level

FMC TDC DMA4 acquisition ready.

FMC TDC timestamps interrupt (DMA5)

HW prefix:	tdc_eic_tdc_dma5
C prefix:	TDC_DMA5
Trigger:	high level

FMC TDC DMA5 acquisition ready.

5.2.7 I2C

Not used.

5.2.8 Mem

5.2.9 Timestamp FIFO

Memory map summary

SW Offset	Type	Name	HW prefix	C prefix
0x0	REG	Delta Timestamp Word 1	tsf_delta1	DELTA1
0x4	REG	Delta Timestamp Word 2	tsf_delta2	DELTA2
0x8	REG	Delta Timestamp Word 3	tsf_delta3	DELTA3
0xc	REG	Channel Offset Word 1	tsf_offset1	OFFSET1
0x10	REG	Channel Offset Word 2	tsf_offset2	OFFSET2
0x14	REG	Channel Offset Word 3	tsf_offset3	OFFSET3
0x18	REG	Control/Status	tsf_csr	CSR
0x1c	FIFOREG	FIFO 'Timestamp FIFO' data output register 0	tsf_fifo_r0	FIFO_R0
0x20	FIFOREG	FIFO 'Timestamp FIFO' data output register 1	tsf_fifo_r1	FIFO_R1
0x24	FIFOREG	FIFO 'Timestamp FIFO' data output register 2	tsf_fifo_r2	FIFO_R2
0x28	FIFOREG	FIFO 'Timestamp FIFO' data output register 3	tsf_fifo_r3	FIFO_R3
0x2c	REG	FIFO 'Timestamp FIFO' control/status register	tsf_fifo_csr	FIFO_CSR

Register description

Delta Timestamp Word 1

HW prefix:	tsf_delta1
HW address:	0x0
SW prefix:	DELTA1
SW offset:	0x0

31	30	29	28	27	26	25	24
DELTA1[31:24]							
23	22	21	20	19	18	17	16
DELTA1[23:16]							
15	14	13	12	11	10	9	8
DELTA1[15:8]							
7	6	5	4	3	2	1	0
DELTA1[7:0]							

- **DELTA1** [*read-only*]: Delta Timestamp Word 1 (TAI cycles, signed)

Delta Timestamp Word 2

HW prefix:	tsf_delta2
HW address:	0x1
SW prefix:	DELTA2
SW offset:	0x4

31	30	29	28	27	26	25	24
DELTA2[31:24]							
23	22	21	20	19	18	17	16
DELTA2[23:16]							
15	14	13	12	11	10	9	8
DELTA2[15:8]							
7	6	5	4	3	2	1	0
DELTA2[7:0]							

- **DELTA2** [*read-only*]: Delta Timestamp Word 2 (8ns ticks, unsigned)

Delta Timestamp Word 3

HW prefix:	tsf_delta3
HW address:	0x2
SW prefix:	DELTA3
SW offset:	0x8

31	30	29	28	27	26	25	24
DELTA3[31:24]							
23	22	21	20	19	18	17	16
DELTA3[23:16]							
15	14	13	12	11	10	9	8
DELTA3[15:8]							
7	6	5	4	3	2	1	0
DELTA3[7:0]							

- **DELTA3** [*read-only*]: Delta Timestamp Word 3 (fractional part, unsigned)

Channel Offset Word 1

HW prefix:	tsf_offset1
HW address:	0x3
SW prefix:	OFFSET1
SW offset:	0xc

31	30	29	28	27	26	25	24
OFFSET1[31:24]							
23	22	21	20	19	18	17	16
OFFSET1[23:16]							
15	14	13	12	11	10	9	8
OFFSET1[15:8]							
7	6	5	4	3	2	1	0
OFFSET1[7:0]							

- **OFFSET1** [*read/write*]: Channel Offset Word 1 (TAI cycles, signed)

Channel Offset Word 2

HW prefix:	tsf_offset2
HW address:	0x4
SW prefix:	OFFSET2
SW offset:	0x10

31	30	29	28	27	26	25	24
OFFSET2[31:24]							
23	22	21	20	19	18	17	16
OFFSET2[23:16]							
15	14	13	12	11	10	9	8
OFFSET2[15:8]							
7	6	5	4	3	2	1	0
OFFSET2[7:0]							

- **OFFSET2** [*read/write*]: Channel Offset Word 2 (8ns ticks, unsigned)

Channel Offset Word 3

HW prefix:	tsf_offset3
HW address:	0x5
SW prefix:	OFFSET3
SW offset:	0x14

31	30	29	28	27	26	25	24
OFFSET3[31:24]							
23	22	21	20	19	18	17	16
OFFSET3[23:16]							
15	14	13	12	11	10	9	8
OFFSET3[15:8]							
7	6	5	4	3	2	1	0
OFFSET3[7:0]							

- **OFFSET3** [*read/write*]: Channel Offset Word 3 (fractional part, unsigned)

Control/Status

HW prefix:	tsf_csr
HW address:	0x6
SW prefix:	CSR
SW offset:	0x18

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	RAW_MODE	DELTA_REF[2:0]			RST_SEQ	DELTA_READ	DELTA_READY

- **DELTA_READY** [*read-only*]: Delta Timestamp Ready
- **DELTA_READ** [*write-only*]: Read Delta Timestamp
- **RST_SEQ** [*write-only*]: Reset Sequence Counter
- **DELTA_REF** [*read/write*]: Delta Timestamp Reference Channel
  Channel (0-4) to take as the reference for the delta timestamps
- **RAW_MODE** [*read/write*]: Raw readout mode
  1: enables readout of raw timestamps

FIFO 'Timestamp FIFO' data output register 0

HW prefix:	tsf_fifo_r0
HW address:	0x7
SW prefix:	FIFO_R0
SW offset:	0x1c

31	30	29	28	27	26	25	24
TS0[31:24]							
23	22	21	20	19	18	17	16
TS0[23:16]							
15	14	13	12	11	10	9	8
TS0[15:8]							
7	6	5	4	3	2	1	0
TS0[7:0]							

- **TS0** [*read-only*]: The timestamp (word 0)

FIFO 'Timestamp FIFO' data output register 1

HW prefix:	tsf_fifo_r1
HW address:	0x8
SW prefix:	FIFO_R1
SW offset:	0x20

31	30	29	28	27	26	25	24
TS1[31:24]							
23	22	21	20	19	18	17	16
TS1[23:16]							
15	14	13	12	11	10	9	8
TS1[15:8]							
7	6	5	4	3	2	1	0
TS1[7:0]							

- **TS1** [*read-only*]: The timestamp (word 1)

FIFO 'Timestamp FIFO' data output register 2

HW prefix:	tsf_fifo_r2
HW address:	0x9
SW prefix:	FIFO_R2
SW offset:	0x24

31	30	29	28	27	26	25	24
TS2[31:24]							
23	22	21	20	19	18	17	16
TS2[23:16]							
15	14	13	12	11	10	9	8
TS2[15:8]							
7	6	5	4	3	2	1	0
TS2[7:0]							

- **TS2** [*read-only*]: The timestamp (word 2)

FIFO 'Timestamp FIFO' data output register 3

HW prefix:	tsf_fifo_r3
HW address:	0xa
SW prefix:	FIFO_R3
SW offset:	0x28

31	30	29	28	27	26	25	24
TS3[31:24]							
23	22	21	20	19	18	17	16
TS3[23:16]							
15	14	13	12	11	10	9	8
TS3[15:8]							
7	6	5	4	3	2	1	0
TS3[7:0]							

- **TS3** [*read-only*]: The timestamp (word 3)

FIFO 'Timestamp FIFO' control/status register

HW prefix:	tsf_fifo_csr
HW address:	0xb
SW prefix:	FIFO_CSR
SW offset:	0x2c

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	CLEAR_BUS	EMPTY	FULL
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	USEDW[5:0]					

- **FULL** [*read-only*]: FIFO full flag
  1: FIFO 'Timestamp FIFO' is full
  0: FIFO is not full
- **EMPTY** [*read-only*]: FIFO empty flag
  1: FIFO 'Timestamp FIFO' is empty
  0: FIFO is not empty
- **CLEAR_BUS** [*write-only*]: FIFO clear
  write 1: clears FIFO 'Timestamp FIFO
  write 0: no effect

- **USEDW** [*read-only*]: FIFO counter
  Number of data records currently being stored in FIFO 'Timestamp FIFO'

5.2.10 Mem DMA

5.2.11 TDC DMA Buffer Control Registers

Memory map summary

SW Offset	Type	Name	HW prefix	C prefix
0x0	REG	Control/Status register	tdc_buf_csr	CSR
0x4	REG	Current buffer base address register	tdc_buf_cur_base	CUR_BASE
0x8	REG	Current buffer base count register	tdc_buf_cur_count	CUR_COUNT
0xc	REG	Current buffer base size/valid flag register	tdc_buf_cur_size	CUR_SIZE
0x10	REG	Next buffer base address register	tdc_buf_next_base	NEXT_BASE
0x14	REG	Next buffer base size/valid flag register	tdc_buf_next_size	NEXT_SIZE

Register description

Control/Status register

HW prefix:	tdc_buf_csr
HW address:	0x0
SW prefix:	CSR
SW offset:	0x0

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
OVERFLOW	DONE	SWITCH_BUFFERS	BURST_SIZE[9:5]				
15	14	13	12	11	10	9	8
BURST_SIZE[4:0]					IRQ_TIMEOUT[9:7]		
7	6	5	4	3	2	1	0
IRQ_TIMEOUT[6:0]							ENABLE

- **ENABLE** [*read/write*]: Enable acquisition
  1: timestamps of the given channel will be sequentially written to the current buffer, provided it's valid (CUR_SIZE.VALID=1)
  0: acquisition off
- **IRQ_TIMEOUT** [*read/write*]: IRQ Timeout (ms)
  Interrupt coalescing timeout in milliseconds. Pick a high enough value to avoid too frequent interrupts and a low enough one to prevent buffer contention. 10 ms should be OK for most of the cases
- **BURST_SIZE** [*read/write*]: Burst size (timestamps)
  Number of timestamps in a single burst to the DDR memory. Default = 16
- **SWITCH_BUFFERS** [*write-only*]: Switch buffers
  write 1: atomically switches the acquisition buffer from the current one (base/size in CUR_BASE/CUR_SIZE) to the next

one (described in NEXT_BASE/NEXT_SIZE registers)
write 0: no action

- **DONE** [*read/write*]: Burst complete
  read 1: the current buffer has been fully committed to the DDR memory after writing 1 to SWITCH_BUFFERS field.
  read 0: still some transfers pending
- **OVERFLOW** [*read/write*]: DMA overflow
  read 1: both the current and the next buffer have been filled with timestamps. Dropping all new incoming TS.

Current buffer base address register

HW prefix:	tdc_buf_cur_base
HW address:	0x1
SW prefix:	CUR_BASE
SW offset:	0x4

31	30	29	28	27	26	25	24
CUR_BASE[31:24]							
23	22	21	20	19	18	17	16
CUR_BASE[23:16]							
15	14	13	12	11	10	9	8
CUR_BASE[15:8]							
7	6	5	4	3	2	1	0
CUR_BASE[7:0]							

- **CUR_BASE** [*read/write*]: Base address
  Base address of the current buffer (in bytes) relative to the DDR3 chip (0 = first word in the memory)

Current buffer base count register

HW prefix:	tdc_buf_cur_count
HW address:	0x2
SW prefix:	CUR_COUNT
SW offset:	0x8

31	30	29	28	27	26	25	24
CUR_COUNT[31:24]							
23	22	21	20	19	18	17	16
CUR_COUNT[23:16]							
15	14	13	12	11	10	9	8
CUR_COUNT[15:8]							
7	6	5	4	3	2	1	0
CUR_COUNT[7:0]							

- **CUR_COUNT** [*read-only*]: Number of data samples
  Number of data samples in the buffer (1 sample = 1 timestamp)

Current buffer base size/valid flag register

HW prefix:	tdc_buf_cur_size
HW address:	0x3
SW prefix:	CUR_SIZE
SW offset:	0xc

31	30	29	28	27	26	25	24
-	VALID	SIZE[29:24]					
23	22	21	20	19	18	17	16
SIZE[23:16]							
15	14	13	12	11	10	9	8
SIZE[15:8]							
7	6	5	4	3	2	1	0
SIZE[7:0]							

- **SIZE** [*read/write*]: Size
  Number of data samples the buffer can hold (1 sample = 1 timestamp)
- **VALID** [*read/write*]: Valid flag
  write 1: indicate that this buffer is ready for acquisition and correctly configured

Next buffer base address register

HW prefix:	tdc_buf_next_base
HW address:	0x4
SW prefix:	NEXT_BASE
SW offset:	0x10

31	30	29	28	27	26	25	24
NEXT_BASE[31:24]							
23	22	21	20	19	18	17	16
NEXT_BASE[23:16]							
15	14	13	12	11	10	9	8
NEXT_BASE[15:8]							
7	6	5	4	3	2	1	0
NEXT_BASE[7:0]							

- **NEXT_BASE** [*read/write*]: Base address

Next buffer base size/valid flag register

HW prefix:	tdc_buf_next_size
HW address:	0x5
SW prefix:	NEXT_SIZE
SW offset:	0x14

31	30	29	28	27	26	25	24
-	VALID	SIZE[29:24]					
23	22	21	20	19	18	17	16
SIZE[23:16]							
15	14	13	12	11	10	9	8
SIZE[15:8]							
7	6	5	4	3	2	1	0
SIZE[7:0]							

- **SIZE** [*read/write*]: Size (in transfers)
- **VALID** [*read/write*]: Valid flag

5.2.12 Mem DMA EIC

5.2.13 GN4124 DMA enhanced interrupt controller

Enhanced interrrupt controller for GN4124 DMA.

Memory map summary

SW Offset	Type	Name	HW prefix	C prefix
0x20	REG	Interrupt disable register	dma_eic_eic_idr	EIC_IDR
0x24	REG	Interrupt enable register	dma_eic_eic_ier	EIC_IER
0x28	REG	Interrupt mask register	dma_eic_eic_imr	EIC_IMR
0x2c	REG	Interrupt status register	dma_eic_eic_isr	EIC_ISR

Register description

Interrupt disable register

HW prefix:	dma_eic_eic_idr
HW address:	0x8
SW prefix:	EIC_IDR
SW offset:	0x20

Writing 1 disables handling of the interrupt associated with corresponding bit. Writin 0 has no effect.

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	-	DMA_ERROR	DMA_DONE

- **DMA_DONE** [*write-only*]: DMA done interrupt
  write 1: disable interrupt 'DMA done interrupt'
  write 0: no effect
- **DMA_ERROR** [*write-only*]: DMA error interrupt
  write 1: disable interrupt 'DMA error interrupt'
  write 0: no effect

Interrupt enable register

HW prefix:	dma_eic_eic_ier
HW address:	0x9
SW prefix:	EIC_IER
SW offset:	0x24

Writing 1 enables handling of the interrupt associated with corresponding bit. Writin 0 has no effect.

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	-	DMA_ERROR	DMA_DONE

- **DMA_DONE** [*write-only*]: DMA done interrupt
  write 1: enable interrupt 'DMA done interrupt'
  write 0: no effect
- **DMA_ERROR** [*write-only*]: DMA error interrupt
  write 1: enable interrupt 'DMA error interrupt'
  write 0: no effect

Interrupt mask register

HW prefix:	dma_eic_eic_imr
HW address:	0xa
SW prefix:	EIC_IMR
SW offset:	0x28

Shows which interrupts are enabled. 1 means that the interrupt associated with the bitfield is enabled

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	-	DMA_ERROR	DMA_DONE

- **DMA_DONE** [*read-only*]: DMA done interrupt
  read 1: interrupt 'DMA done interrupt' is enabled
  read 0: interrupt 'DMA done interrupt' is disabled
- **DMA_ERROR** [*read-only*]: DMA error interrupt
  read 1: interrupt 'DMA error interrupt' is enabled
  read 0: interrupt 'DMA error interrupt' is disabled

Interrupt status register

HW prefix:	dma_eic_eic_isr
HW address:	0xb
SW prefix:	EIC_ISR
SW offset:	0x2c

Each bit represents the state of corresponding interrupt. 1 means the interrupt is pending. Writing 1 to a bit clears the corresponding interrupt. Writing 0 has no effect.

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	-	DMA_ERROR	DMA_DONE

- **DMA_DONE** [*read/write*]: DMA done interrupt
  read 1: interrupt 'DMA done interrupt' is pending

read 0: interrupt not pending
write 1: clear interrupt 'DMA done interrupt'
write 0: no effect

- **DMA_ERROR** [*read/write*]: DMA error interrupt
  read 1: interrupt 'DMA error interrupt' is pending
  read 0: interrupt not pending
  write 1: clear interrupt 'DMA error interrupt'
  write 0: no effect

Interrupts

DMA done interrupt

HW prefix:	dma_eic_dma_done
C prefix:	DMA_DONE
Trigger:	rising edge

DMA done interrupt line (rising edge sensitive).

DMA error interrupt

HW prefix:	dma_eic_dma_error
C prefix:	DMA_ERROR
Trigger:	rising edge

DMA error interrupt line (rising edge sensitive).

INDEX