FMC TDC User's Manual

July 2013

FMC TDC 1ns-5cha hardware and software manual

CERN BE-CO-HT / Tomasz Wlostowski, Alessandro Rubini

Table of Contents

Introduction

This is the user manual for the *FmcTdc1ns5cha* board developed on `ohwr.org`, referenced futher as the *FmcTdc*. The manual is heavily based on the documentation of the *Fine Delay* card, written by Alessandro Rubini.

1 Repositories and Releases

The code and documentation are distributed in the following places:

`http://www.ohwr.org/projects/fmc-tdc-sw/documents`
: This place hosts the pdf documentation for official releases.

`http://www.ohwr.org/projects/fmc-tdc-sw/files`
: Here we place the *.tar.gz* file for every release, including the *git* tree and compiled documentation (for those who lack TeX).

`git://ohwr.org/fmc-projects/fmc-tdc/fmc-tdc-sw.git`
: Read-only repository for the software and documentation.

`git@ohwr.org:fmc-projects/fmc-tdc/fmc-tdc-sw.git`
: Read-write repositories, for those authorized.

Note: If you got this from the repository (as opposed to a named *tar.gz* or *pdf* file) it may happen that you are looking at a later commit than the release this manual claims to document. It is a fact of life that developers forget to re-read and fix documentation while updating the code. In that case, please run "`git describe HEAD`" to ensure where you are.

2 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 a SVEC VME64x FMC carrier (supplied with one or two *FmcTdc*s),
- 50-ohm cables with 1-pin LEMO 00 plugs for connecting the I/O signals,
- Any Linux (kernel 2.6 or 3.0+) distribution.

Note: The software has been developed on Linux-3.11, and supports legacy kernels down to 2.6.24 (CERN RT-patched kernel).

2.2 Mechanical/Environmental

Mechanical and environmental specs:

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

2.3 Electrical

Inputs/Outputs:

- 5 trigger inputs (LEMO 00),
- 6 LEDs: 5 for indicating input termination, 1 as a general-purpose status 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). 50 Ohm termination is indicated by a corresponding LED in the front panel,
- 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).
- Typical current consumption: FIXME (P12V0) + FIXME (P3V3).
- Power dissipation: [fixme: Eva] W.

2.4 Timing

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.

Input timing:

- Minimum pulse width: t_{IW} = 100 ns. Pulses below 100 ns are rejected. Width checking is done in software by subtracting rising and falling edge timestamps.
- Minimum pulse spacing: 100 ns.
- Only rising edges are time tagged.
- TDC accuracy: 700 ps peak-peak (six sigma)
- TDC precision: 500 ps peak-peak (six sigma).
- TDC resolution: 81 ps.

3 Driver Features

This driver is based on *ZIO* and *fmc-bus*. The features it provides are:

- Setting up the board.
- Reading time.
- The obvious: reading timestamps of incoming pulses.
- User-defined input offsets.
- Readout of card temperature.
- Controlling channel termination.
- Globally enabling/disabling timestamp acquisition.

The features that are planned, but are **not supported** in this release are:

- White Rabbit synchronization

For each feature offered the driver (and documentation) the driver tries to offer the following items; sometimes however one of them is missing for a specific driver functionality, if we don't consider it important enough.

- A description how does the feature work.
- A C-language API to access the feature with data structures.
- An example program based on that API.

4 Installation

This driver depends on three other modules (three `ohwr.org` packages), as well as the Linux kernel. Also, it must talk to a specific FPGA binary file running in the device.

4.1 Gateware Dependencies

This version of the driver has been developed to run with the FPGA binary included in the package as `binaries/[carrier]-fmc-tdc.bin`. The `[carrier]` prefix denotes the type of FMC carrier the gateware is for - that is, `spec` or `svec`. These binary files are included in the software release package. They are also available directly in the *Files* tab on the OHWR project website.

If the gateware is updated, I'll take care to always include in this package the exact binary the software is developed and verified against.

4.2 Gateware Installation

To install the FPGA image in the target system, please follow the instructions in the documentation of *spec-sw* or *svec-sw*. To summarize, you'll need to place the `.bin` file, properly renamed, in */lib/firmware* or a subdirectory thereof. The default name used by this driver is `fmc/spec-fmc-tdc.bin` or `fmc/svec-fmc-tdc.bin`.

If you have several *FMC TDC* cards in the same host, you can load different binaries in different cards, using appropriate module parameters.

The following example commands are sufficient for most users:

```
$ sudo mkdir -p /lib/firmware/fmc
$ sudo cp spec-fmc-tdc.bin svec-fmc-tdc.bin /lib/firmware/fmc
```

4.3 Software Dependencies

The kernel versions I am using during development is 3.11. Everything used here is known to build with all versions since 2.6.32 and up to 3.11. Note that the *FmcTdc* driver will compile older kernels (such as 2.6.24-rt that is used at CERN), but the modules it depends on will have to be built from backport branches.

The driver, then, is based on the ZIO framework, available from `ohwr.org`. I'm developing with the `v1.0-fixes` branch of the framework. Again, this commit of ZIO is known to work in the kernel range 2.6.32..3.11 and a backport to 2.6.24 is available.

The FMC mezzanine is supported by means of the *fmc-bus* software project. Such support used to be part of the *spec-sw* package, but is not a project of its own. This *FmcTdc* kernel module registers as a *driver* for the FMC bus abstraction, and is verified with version `v2013-05.1` of the FMC package. The same kernel range applies.

Both packages (ZIO and *fmc-bus*) 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.

The carrier driver is not strictly related to this package, but *FmcTdc* is released against version `v2013-04` of *spec-sw* and version [fixme] of *svec-sw*.

Unfortunately, all the packages are moving fast: we are approaching a stable and long-lasting status but we are not there yet. Please stick to the released versions named in this section, unless you are involved in development.

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.

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-tdc.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.

After installation, your carrier driver should load automatically (for example, the PCI bus will load `spec.ko`), but `fmc-tdc.ko` must be loaded manually, because support for automatic loading is not yet in place. The suggested command is one or the other of the following two:

```
modprobe fmc-tdc [<parameter> ...]         # after make install
insmod kernel/fmc-tdc.ko [<parameter> ...]  # if not installed
```

Available module parameters are described in Section 4.5 [Module Parameters], page 5. Unless you customized or want to customize one of the three related packages, you can skip the rest of this section.

Note that in case of the VME version, one must configure VME base addresses/LUNs through parameters of the SVEC driver. A sample set of commands is below:

```
modprobe fmc-tdc [<parameter> ...]         # after make install
modprobe svec slot=4 vmebase=0xa0000000 lun=0 vector=0x86
```

It assumes a SVEC with A32 mapping at `0xa0000000`, identified as card `0` in the system and residing in slot `4` of the VME crate.

In order to compile *FmcTdc* against a specific repository of one of the related packages, ignoring the local *submodule* you can use one or more of the following environment variables:

`ZIO`
`FMC_BUS`

: The top-level directory of the repository checkout of each package. Most users won't need to set them, as the Makefiles point them to the proper place by default.

If any of the above is set, headers and dependencies for the respective package are taken from the chosen directory. If you `make prereq_install` with any of these variables set, they are be used to know where to install from, instead of using local submodules.

4.5 Module Parameters

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

The following parameters are used:

`verbose=`

: The parameter defaults to 0. If set, it enables more diagnostic messages during probe (you may find it is not used, but it is left in to be useful during further development, and avoid compile-time changes like use of `#ifdef DEBUG`).

`poll_interval=`

: The period of the buffer polling timer. The timer is used to poll for input events on the SVEC card, whose software does not support interrupts yet. The default interval is 10 milliseconds. You may want to use the timer while porting to a different carrier, before sorting out IRQ issues.

`buffer_size`

: Numer of entries in the software timestamp buffer (independent buffers per channel). Defaults to 8192.

`show_sdb`

: Defaults to 0. If enabled, the driver will print out the contents of the SDB device tree in the gateware. Reserved for debugging purposes.

The module also uses the two parameters provided by the *fmc* framework:

`busid=`

: A list of bus identifiers the driver will accept to driver. Other identifiers will lead to a failure in the *probe* function. The meaning of the identifiers is carrier-specific; the SPEC uses the bus number and *devfn*, where the latter is most likely zero.

`gateware=`

A list of gateware file names. The names passed are made to match the *busid* parameters, in the same order. This means that you can't make the driver load a different gateware file without passing the respective *busid*. Actually, to change the gateware for all boards, you may just replace the file in `/lib/firmware`. (Maybe I'll add an option to change the name at load time for all boards).

For example, this host has one SPEC cards:

```
# lspci | grep CERN
01:00.0 Non-VGA unclassified device: CERN/ECP/EDU Device 018d (rev 03)
```

Installing the `fmc-tdc` driver should show the following output:

```
# insmod fmc-tdc.ko
[321272.596475] spec 0000:01:00.0: reprogramming with fmc/spec-fmc-tdc.bin
[321272.791378] spec 0000:01:00.0: FPGA programming successful
[321272.791478] spec 0000:01:00.0: Gateware successfully loaded
[321272.791483] fmc_tdc FmcTdc1ns5cha-0100: ft_spec_reset: resetting TDC core through Gennum.
[321275.798831] fmc_tdc FmcTdc1ns5cha-0100: calib: zero_offset[0] = 0
[321275.798835] fmc_tdc FmcTdc1ns5cha-0100: calib: zero_offset[1] = 86
[321275.798837] fmc_tdc FmcTdc1ns5cha-0100: calib: zero_offset[2] = 609
[321275.798840] fmc_tdc FmcTdc1ns5cha-0100: calib: zero_offset[3] = 572
[321275.798842] fmc_tdc FmcTdc1ns5cha-0100: calib: zero_offset[4] = 335
[321275.798844] fmc_tdc FmcTdc1ns5cha-0100: calib: vcxo_default_tune 43343
[321275.814487] fmc_tdc FmcTdc1ns5cha-0100: ft_acam_init: ACAM initialization OK.
[321276.580679] fmc_tdc FmcTdc1ns5cha-0100: ft_read_temp: Scratchpad:
[321276.580683] e3:02:4b:46:7f:ff:0d:10:c0
[321276.580692] fmc_tdc FmcTdc1ns5cha-0100: ft_read_temp: Temperature 0x2e3 (12 bits: 46.187)
```

5 Source Code Conventions

This is a random list of conventions I use in this package

- All internal symbols in the driver begin with `ft_` (excluding local variables like i and similar stuff). So you know if something is local or comes from the kernel.
- All library functions and public data begin with `fmctdc_`.
- The board passed as a library token (`struct fmctdc_board`) is opaque, so the user doesn't access it. Internally it is called `userb` because `b` is the real one being used. If you need to access library internals from a user file just include `fmctdc-lib-private.h` after including `fmctdc-lib.h`.
- The driver header is called `fmc-tdc.h` while the user one is `fmctdc-lib.h`. The latter includes the former, which user programs should not refer to.
- The *test* contains a set of example programs for the library.

6 API Overview

In this chapter we will not discuss about the details of the API; for this purpose please generate (and read) the doxygen documentation:

```
cd doc
make doxygen
```

Here we will present generic concept behind the API, all implementation details are available on the doxygen documentation.

6.1 Time-Stamp modes

The driver provides two time-stamp modes that you can configure for each channel:

- base time
- difference time

The standard mode is the *base time* mode. When this mode is enabled for a given channel, the provided time-stamps will be a pure time-stamp according to the TDC internal base-time. You can change the internal base-time of a TDC board when the acquisition is off.

The *difference time* mode can be enabled by assigning a channel reference to a given channel (target). When you assing a channel reference to a channel the time-stamps produced by the driver will be a time difference between the pulse on the target channel and the last pulse on the reference channel. In order to disable the difference mode, and go back to the base time mode, you must remove the channel reference.

7 Example programs

7.1 Initialization and Cleanup

The sample program *fmctdc-list* lists the boards currently on the system, using *fmctdc_init*:

```
# ./fmctdc-list
Found 1 board(s):
0100, /dev/zio/ft-0100, /sys/bus/zio/devices/ft-0100
```

7.2 Time Management

The program *fmctdc-board* is a command-line front-end to the library, to validate the library works as expected. The first parameter is the board bus ID, the second one is the command, the third one is the new seconds counter value:

```
# ./fmctdc-time 0100 get
Current TAI time is 813.000000000 s
# ./fmctdc-time 0100 set 10000
# ./fmctdc-time 0100 get
Current TAI time is 10001.000000000 s
# ./fmctdc-time 0100 host
# ./fmctdc-time 0100 get
Current TAI time is 1376987950.000000000 s
```

7.3 Input Configuration

The example program *fmctdc-term* demonstrates use of the function. It just enables or disables the 50-ohm resistor. The effect is usually verifiable by hooking a scope to the input signal:

```
# ./fmctdc-term 0100
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
channel 5: 50 Ohm termination is off
# ./fmctdc-term 0100 1 on
channel 1: 50 Ohm termination is on
```

The other example, *fmctdc-acquisition*, works as follows:

```
# ./fmctdc-acquisition 0100 off
# ./fmctdc-acquisition 0100
board 0100: acquisition is off
```

7.4 Reading Input Time-stamps

There is an example program provided, called *fmctdc-read*:

Calling *fmctdc-read* with just the board ID will keep reading all timestamps from all channels. One can limit the number of samples to be read by using `-s` parameter.

```
# fmctdc-read 0100
channel 1 seq 1699         ts 1376988979.899,210,574,710 ps
channel 3 seq 11           ts 1376988979.899,210,599,375 ps
channel 1 seq 1700         ts 1376988980.055,610,386,324 ps
channel 3 seq 12           ts 1376988980.055,610,410,908 ps
(...)
```

The program also demonstrates the non-blocking mode, which can be enabled by `-n` parameter. Unfortunately, even if there are samples pending, *read* will only return one of them, because the ZIO device will only see the next sample slightly after returning the previous one. This is a buffering problem with our use of ZIO. Therefore, calling *read* in non-blocking mode will only return the first pending sample from each channel.

The *read* utility can also dump hardware (WR) timestamps instead of converting them to seconds. It is also possible to specify the range of channels to be read:

```
# ./fmctdc-read -w 0100 1
channel 1 seq 1981         ts 1376989024:000548630:2712
channel 1 seq 1982         ts 1376989024:020097955:1262
channel 1 seq 1983         ts 1376989024:039649072:3338
```

Note that if the input pulses come too often, the timestamp buffers will overflow. This is indicated by making a gap in the sequence ID. Due to gateware issues, it is **not possible** to determine the exact number of lost samples by subtracting the sequence IDs.

7.5 Setting user input offsets

It is possible to add an user-defined offset to all timestamps coming to a particular channel, for example to compensate for cabling delay. There is no dedicated API function for that purpose, it may be added in future relases. Setting the offset currently can be done through ZIO `sysfs` parameters. For example:

```
# echo 10000 > /sys/bus/zio/devices/ft-0100/ft-ch1/user-offset
```

will result with the driver adding extra 10 ns (= 10000 ps) to each timestamp coming to channel 1 of the card `0100`.

8 Known Bugs and Missing Features

This package is still work in progress, and unfortunately the same applies to the packages it depends on – *zio* and *fmc-bus*.

8.1 Bugs in Related Packages

The current package set (i.e., *zio*, *fmc-bus* and this one) has the following known issues exposed by *fine-delay*:

- The auto-loading of *fmc* modules is not yet working:
- The *user* trigger of ZIO is really user-driven, so the driver can't push stuff to the buffer until asked to. Also, a related buglet prevents to return data immediately when asked. This will be fixed, but it currently results in the *read* function only returning one sample, and an immediately-following non-blocking *read* will say nothing is there, yet.

8.2 Bugs in This Package

This is the list of known bugs and missing features over what hardware allows:

- Calibration information in the EEPROM is not verified, as its the format does not include any means of verification (hash, checksum, etc.)
- Purging the buffers by disabling acquisition sometimes leaves a single old sample in the ZIO buffer. This is probably due to a bug in ZIO.

8.3 Gateware issues

This is the list of known gateware issues, that sometimes affected the architecture and features of the driver.

- Time setting is possible only when acquisition is disabled.
- No coarse time counter setting possible.
- Seconds counter has 32 bits. This may bring a surprise in 2038 if the time is set according to the Unix epoch.
- No White Rabbit support.
- Acquisition can be only enabled/disabled globally for all channels.
- No support for hardware sequence numbers. This, combined with the HW timestamp buffer shared between all channels, makes it impossible to count the number of lost samples if the HW buffer overflows.
- Pulse width checking has to be done in software.
- No possibility to check FMC's PLL status on the SPEC (if the mezznanine is broken, the OS will freeze without reporting any error).
- SVEC version must have two mezzanines inserted, even if only one of them is used. This is due to clocking internal Wishbone bus directly from FMC PLLs. Both of them need to be running before the driver can read the SDB tree without dropping an error and/or crashing the machine.

8.4 Wish List

Other less important issues may be dealt with over time, but are not urgent as I write this:

- The driver should register its own ZIO trigger, or use the new attribute for "greedy-input" planned in new versions of ZIO (thank you Federico). Currently there's no buffering and reading is slower than it could be.
- White Rabbit support, as in the *Fine Delay*.

9 Troubleshooting

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

9.1 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`.

9.2 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
```

9.3 Version Mismatch

The *fmctdc* library may report a version mismatch like this:

```
spusa# ./lib/fmctdc-time  get
fmctdc_init: version mismatch, lib(1) != drv(2)
./lib/fmctdc-time: fmctdc_init(): Input/output error
```

This reports a difference in the way ZIO attributes are laid out, so user space may exchange wrong data in the ZIO control block, or may try to access inexistent files in */sys*. I suggest recompiling both the kernel driver and user space from a single release of the source package.