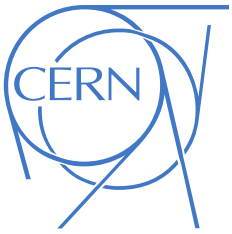
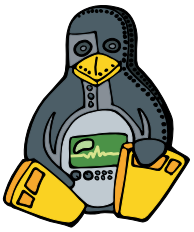
FmcAdc100m14b4cha Firmware Guide

April 2013 - Release 1.1
For Simple PCIe FMC Carrier (SPEC) only





Matthieu Cattin (CERN)

Table of Contents

Introduction

This document describes the hdl (firmware or gateware) developed to support the FmcAdc100m14b4cha (later refered to as fmc-adc) mezzanine card on the SPEC[1] carrier card. The firmware architecture is described in detail. The configuration and operation of the firmware is also explained. On the other hand, this manual is not intended to provide information about the software used to control the fmc-adc board.

1 Repositories and Releases

This project is hosted on the Open Hardware repository at the following link: http://www.ohwr.org/projects/fmc-adc-100m14b4cha

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

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

`File`[3] contains the `.bin` FPGA binary file for every official release.

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

On the repository the official releases have a tag named `spec-fmc-adc-v#maj.#min` where `#maj` represent the major release version of the firmware and `#min` the minor one (e.g `spec-fmc-adc-v1.2`). The released FPGA binary files follow the same naming convention.

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.

1.1 Software Support

For information on the fmc-adc Linux software support, please refer to the following project: http://www.ohwr.org/projects/fmc-adc-100m14b4cha-sw

As a general rule, a new minor version of the firmware, for a given major version, should be backward compatible. If the interface with the driver changes, the major version should be incremented. It means that driver versions 1.x should work with any firmware version 1.x. But the driver version 2.0 might not work with the firmware version 1.1.

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

2 About Source Code

2.1 Build from Sources

The fmc-adc hdl design make use of the `hdlmake`[1] 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.

1. Install `hdlmake`.
2. Get fmc-adc hdl sources.
   `git clone git://ohwr.org/fmc-projects/fmc-adc-100m14b4cha.git <src_dir>`
3. Goto the synthesis directory.
   `cd <src_dir/hdl/spec/syn/>`
4. Fetch the dependencies.
   `hdlmake -f`
5. Generate an ISE project file.
   `hdlmake --ise-proj`
   This will generate a basic ISE project file with default settings. If non-default setting is needed (e.g. binary bitstream output file .bin), the project file must be opened using ISE project navigator GUI and the setting changed manually.
6. Generate a synthesis Makefile.
   `hdlmake --make-ise`
7. Check that all dependencies are fetched.
   `hdlmake --list`
8. Synthesis, place and route.
   `make`

2.2 Source Code Organisation

‘`hdl/adc/rtl/`’
: ADC specific hdl sources.

‘`hdl/adc/wb_gen/`’
: ADC specific `wbgen2` sources, html documentation and C header file.

‘`hdl/spec/rtl/`’
: SPEC carrier related hdl sources.

‘`hdl/spec/wb_gen/`’
: SPEC carrier related `wbgen2` sources, html documentation and C header file.

‘`hdl/spec/ip_cores/`’
: Location of fetched and generated hdl cores and libraries.

‘`hdl/spec/syn/`’
: Synthesis directory for SPEC carrier. This is where the synthesis top manifest and the ISE project are stored. For each release, the synthesis, place&route and timing reports are also saved here.

‘`hdl/spec/sim/`’
: SPEC carrier related simulation files and testbenches.

[1] http://www.ohwr.org/projects/hdl-make

‘`hdl/spec/chipscope/`’
SPEC carrier related Chipscope projects used for debug purpose.

It could happen that a hdl source directory contains extra source files that are not used in the current firmware release. In order to identify the source files used in a given release, refer to the ‘`Manifest.py`’ files.

2.3 Dependencies

The fmc-adc firmware depends on the following hdl cores and libraries:

gn4124-core
`repo : git://ohwr.org/hdl-core-lib/gn4124-core.git`
`branch: master`

ddr3-sp6-core
`repo : git://ohwr.org/hdl-core-lib/ddr3-sp6-core`
`branch: spec_bank3_64b_32b`

general-cores
`repo : git://ohwr.org/hdl-core-lib/general-cores.git`
`branch: sdb_extension`

3 Architecture

This chapter describes the internal blocks of the FPGA. All blocks (except the memory controller) are connected to the PCIe bridge interface using a Wishbone bus. The DDR memory can only be access through DMA. The Figure 3.1 illustrates the FPGA architecture. The peripherals connected to each block are also shown. A crossbar from the general-cores[1] library is used to map the Wishbone slaves in the BAR 0 address space. The Table 3.1 shows the Wishbone slaves mapping.

Byte offset	Core	Library	Description
0x0000	sdb_rom	general-cores	SDB records
0x1000	gn4124_core	gn4124-core	DMA controller
0x1100	-	-	Carrier SPI master[2]
0x1200	xwb_onewire_master	general-cores	Carrier 1-wire master
0x1300	carrier_csr	fmc-adc-100m14b4cha	Carrier control and status
0x1400	utc_core	fmc-adc-100m14b4cha	Time-tagging core
0x1500	irq_controller	fmc-adc-100m14b4cha	Interrupt controller
0x1600	xwb_i2c_master	general-cores	Mezzanine system I2C master
0x1700	xwb_spi	general-cores	Mezzanine SPI master
0x1800	xwb_i2c_master	general-cores	Mezzanine I2C master
0x1900	fmc_adc_100Ms_core	fmc-adc-100m14b4cha	ADC core
0x1A00	xwb_onewire_master	general-cores	Mezzanine 1-wire master

Table 3.1: Wishbone bus memory mapping (BAR 0).

The Wishbone crossbar also implements SDB[3] records. Those records describe the Wishbone slaves and their mapping on the bus. The SDB records ROM must be located at offset 0x0. In order to identify the firmware, SDB meta-information records are used. The 'Integration', 'Top module repository url' and 'Synthesis tool information' meta-information records are used in the design. Below is a description of the fields and their content in the fmc-adc design.

Integration

vendor_id = 0x0000CE42 (CERN vendor ID)
device_id = 0x47C786A2 (echo "spec_fmc-adc-100m14b4cha"|md5sum|cut -c1-8)
version = [31:16]=major, [15:0]=minor, bcd encoded
date = release date, format yyyymmdd
name = "spec_fmcadc100m14b"

Top module repository url

repo_url = "git://ohwr.org/fmc-projects/fmc-adc-100m14b4cha.git"

Synthesis tool information

syn_module_name = "spec_top_fmc_adc"
syn_commit_id = git log -1 –format="%H" | cut -c1-32
syn_tool_name = "ISE"
syn_tool_version = bcd encoded synthesis tool version
syn_date = synthesis date, format yyyymmdd
syn_username = "mcattin" (synthesised by)

[1] http://www.ohwr.org/projects/general-cores
[2] Not implemented.
[3] http://www.ohwr.org/projects/fpga-config-space

Note that some of the cores from the general-cores library are based on cores from OpenCores[4]. Therefore, the documentation for those cores is hosted on the OpenCores website.

The register description for the cores for the carrier control and status, the time-tagging core, the interrupt controller and the ADC core can be found in annexe (See [ADC Core Registers], page 22, [Interrupt Controller Registers], page 31, [Time-tagging Core Registers], page 32 and [Carrier Registers], page 37). The registers for those cores have been generated using `wbgen2`[5].

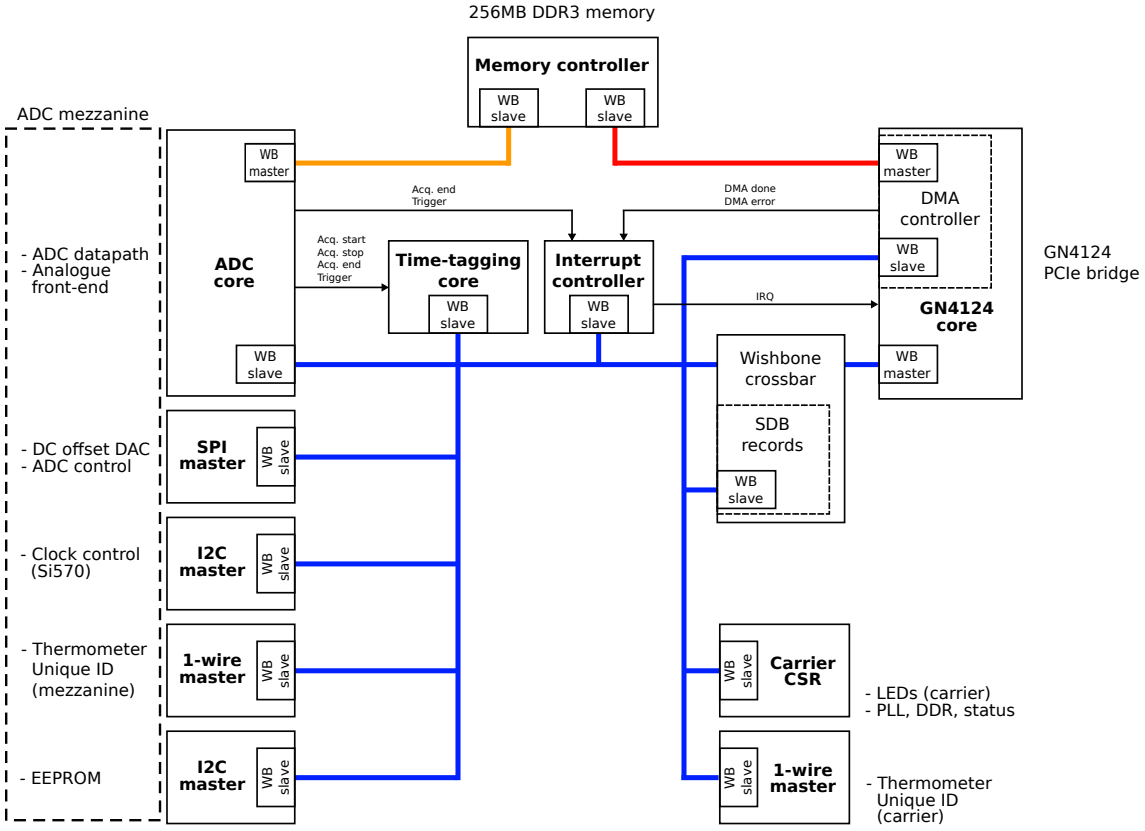


Figure 3.1: FPGA firmware architecture block diargam.

There are three different Wishbone bus in the design.

Mapped WB bus (blue)
: This bus connects all the peripheral to the GN4142 core.
Data: 32-bit, address: 32-bit (word aligned), clock: system clock (125MHz).

ADC core to memory controller (orange)
: This bus is used to write samples from the ADC core to the DDR memory.
Data: 64-bit, address: 32-bit (word aligned), clock: system clock (125MHz).

Memory controller to GN4124 core (red)
: This bus is used to read samples from the DDR memory.
Data: 32-bit, address: 32-bit (word aligned), clock: system clock (125MHz).

[4] http://opencores.org/
[5] http://www.ohwr.org/projects/wishbone-gen

3.1 Clock Domains

The fmc-adc design has four different clock domains. They are listed in the followig table.

Name	Description	Frequency	Source
`sys_clk_125`	Main system clock	125.00 MHz	20MHz TCXO (carrier)
`ddr_clk`	DDR interface clock	333.33 MHz	20MHz TCXO (carrier)
`fs_clk`	Sampling clock	100.00 MHz	400MHz LTC2174 (mezzanine)
`serdes_clk`	ADC data de-serialiser clock	800.00 MHz	400MHz LTC2174 (mezzanine)
`p2l_clk`	Local bus clock	200.00 MHz	200MHz GN4124 (carrier)

> **Note:** By default, the sampling clock is 100MHz. But it can be changed to any frequency from 10MHz to 105MHz. The lower bound is defined by the Si570 programmable oscillator. While the upper bound is limited by the LTC2174 ADC.

> **Note:** The Si570 clock output is connected to the LTC2174 ADC. Then the data clock (DCO) output of the LTC2174 is connected to the FPGA. The data clock is four times the sampling clock. The sampling clock (`fs_clk`) and the ADC data de-serialiser clock (`serdes_clk`) are derived from the data clock using a PLL (internal to the FPGA).

3.2 GN4124 Core

This block is the interface between the GN4124[1] local bus and the other blocks in the FPGA. The GN4124 is a four lane PCI Express Generation 1.1 bridge. In addition to th PHY, it also contains the data link and transaction layers. The GN4124 bridge is used to access the FPGA registers, but also to generate MSI interrupts and re-program the FPGA. The BAR 4 (Base Address Register) allows access to the GN4124 internal registers. The BAR 0 is connected to the local bus and therefore allows access to the FPGA.

The GN4124 core is made of a local bus interface with the GN4124 chip, a Wishbone bus master mapped to BAR0 and a DMA controller. The DMA controller has two Wishbone ports, a Wishbone slave to configure the DMA controller and a Wishbone master to access the DDR memory. The GN4124 Wishbone interfaces (masters and slave) are 32-bit data width and 32-bit word aligned addresses.

> **Note:** It is not possible to insert an address converter (for non-interleaved data read) between the GN4124 core and the memory controller. Because the DDR memory access is not efficiant when reading non-consecutive addresses.

3.3 Carrier Control and Status

This block contains control and status registers related to the carrier board. A first register allows to readout the carrier PCB revision and carrier type. Another register signals the presence of a mezzanine in the FMC slot, gives the status of the local bus and system PLLs and indicates the DDR memory controller calibration state. The last register of this block allows to control the carrier's LEDs on the front panel. There is on red and one green LED.

> **Note:** The "Carrier Type" field is used only for test purpose. The carrier board identification is done through the PCI Express vendor and device ID.

[1] PCI Express bridge from Semtech (formerly Gennum)

3.4 Carrier 1-wire Master

This 1-wire master controls the DS18B20 thermometer chip located on the carrier board. This chip also contains a unique 64-bit identifier. This block is based on an OpenCores design.

This block is clocked by the system clock (125 MHz). Therefore the dividers configuration are `CDR_N=624` and `CDR_O=124`.

```
CDR_N = f_sys * 5E-6 - 1
CDR_O = f_sys * 1E-6 - 1
```

http://opencores.org/project,sockit_owm

3.5 Carrier SPI Master

The carrier SPI master is not implemented. It is meant to control DACs connected to VCXO for White Rabbit[2] applications.

3.6 Memory Controller

The memory controller block is the interface between the 256MB DDR memory located on the SPEC board and the other blocks in the FPGA. It is basically a MCB core (Memory Controller Block) generated with Xilinx CoreGen and an additional wrapper implementing two Wishbone slave interfaces. One of the Wishbone slave interface is connected to the ADC core. The other Wishbone slave interface is connected to the DMA Wishbone master of the GN4124 core.

WB Slave	WB Master	Data width	Access type
0	ADC core	64-bit	Write only
1	GN4124 core	32-bit	Read/write

The memory controller side connected to the chip is 16-bit DDR data bus, clocked at 333.33 MHz. This gives a maximum bandwidth of 1333.33 MB/s. Each of the four channel is 200 MB/s, for a total of 800 MB/s. In the current design, the two Wishbone port have the same priority. The arbitration is done with a simple round-robin. Therefore, the samples stored in the DDR memory cannot be read during an acquisition.

http://www.ohwr.org/projects/ddr3-sp6-core

http://www.xilinx.com/support/documentation/user_guides/ug388.pdf

3.7 Interrupt Controller

The interrupt controller purpose is to concentrate several interrupt source into one interrupt request line. It has four interrupt inputs and one interrupt request output. It also have one interrupt enable mask register and one interrupt source register. Each time a valid rising edge is detected on one of the inputs, a pulse is generated on the interrupt request output. The interrupt request output is connected to the GPIO 8 of the GN4124 chip. Note that the GN4124 must be configured to generate a MSI when a rising edge is detected on GPIO 8. A rising edge is valid if the corresponding bit in the interrupt enable mask register is set. When a valid rising edge is detected, the corresponding bit in the interrupt source register is set as well. This indicates to the host which source caused the interrput. To clear a bit in the interrupt source register, a '1' must be written to it.

There are four interrupt sources in the design. They are listed below.

[2] http://www.ohwr.org/projects/white-rabbit

DMA done
: This interrupt signals the end of a DMA transfer.

DMA error
: This interrupt signals an error in a DMA transfer.

Trigger
: This interrupt signals that a valid trigger arrived while the acquisition state machine was in the `WAIT_TRIG` state.

Acquisition end
: This interrupt signals the end of an acquisition. In case of multi-shot acquisition, it occurs at the end of the last shot.

3.8 Time-tagging Core

This block allows time-tagging of important events in the ADC core. It is based on a seconds counter and a 125MHz system clock ticks counter. Those two counters are accessible in read/write mode via registers. To time-tag the events, the ADC core sends pulses to the time-tagging core. The following events are time-tagged; trigger, acquisition start, acquisition stop and acquisition end.

> **Note:** In this release, the meta-data register is NOT used, set to zero.
>
> **Note:** If during an acqusition no stop command is issued (normal case), the acquisition time-tag is not updated.

3.9 ADC Core

The ADC core is the main block of the design. On the mezzanine interface side, it takes a data flow from the LTC2174 ADC chip, an external trigger and controls the analogue switches to select the input range or calibration mode. On the internal interface side, it has a Wishbone master to write data to the DDR memory controller. It also has a Wishbone slave to access the core's control and status regiters. In addition, it outputs the following event as pulses:

- Trigger
- Acquisition start
- Acquisition stop
- Acquisition end

The internal detailed functionning of this block is described further in the document(See Chapter 4 [Configuration], page 10, Chapter 6 [Acquisition], page 16 and Chapter 5 [Calibration], page 14).

3.10 Mezzanine SPI Master

This SPI master controls the LTC2174 ADC and the four MAX5442 offset DACs. The following table shows how the peripherals are wired to the core. This block is based on an OpenCores design.

SPI slave select	Peripheral
0	LTC2174 ADC
1	MAX5442 DAC for channel 1
2	MAX5442 DAC for channel 2
3	MAX5442 DAC for channel 3
4	MAX5442 DAC for channel 4

This block is clocked by the system clock (125 MHz). Therefore for a SCLK of ~620 kHz, the divider configuration is `DIVIDER=100`.

```
f_sclk = f_sys / ((DIVIDER+1) * 2)
```

http://opencores.org/project,spi

http://cds.linear.com/docs/en/datasheet/21754314fa.pdf

http://datasheets.maximintegrated.com/en/ds/MAX5441-MAX5444.pdf

3.11 Mezzanine 1-wire Master

This 1-wire master controls the DS18B20 thermometer chip located on the mezzanine board. This chip also contains a unique 64-bit identifier. This block is based on an OpenCores design.

This block is clocked by the system clock (125 MHz). Therefore the dividers configuration are `CDR_N=624` and `CDR_O=124`.

http://opencores.org/project,sockit_owm

http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

3.12 Mezzanine I2C Master

This I2C master controls the Si570 programmable oscillator chip located on the mezzanine board. This chip is used to produce the ADC sampling clock. This block is based on an OpenCores design.

I2C slave address	Peripheral
0x55	Si570 programmable oscillator

This block is clocked by the system clock (125 MHz). Therefore for a SCL clock of 100 kHz, the prescaler configuration is `PRESCALER=249`.

```
PRESCALER = f_sys / (5 * f_scl) - 1
```

http://opencores.org/project,i2c

https://www.silabs.com/Support%20Documents/TechnicalDocs/si570.pdf

3.13 Mezzanine System Management I2C Master

This I2C master access the 24AA64 64Kb EEPROM memory chip located on the mezzanine board. This memory is mandatory as specified in the FMC standard (VITA 57.1). It is connected to the system management I2C bus, also specified in the FMC standard. This block is based on an OpenCores design.

I2C slave address	Peripheral
0x50	24AA64 64Kb EEPROM memory

This block is clocked by the system clock (125 MHz). Therefore for a SCL clock of 100 kHz, the prescaler configuration is `PRESCALER=249`.

```
PRESCALER = f_sys / (5 * f_scl) - 1
```

http://opencores.org/project,i2c

http://ww1.microchip.com/downloads/en/devicedoc/21189f.pdf

4 Configuration

The Figure 4.1 is a block diagram of the ADC core part in the sampling clock domain. It contains a ADC data stream de-serialiser, an offset and gain correction block (for ADC data), an under-sampling block and a trigger unit. The four channels data and the trigger signal are synchronised to the system clock domain using a FIFO. The configuration signals coming from registers in the system clock domain are synchronised to the sampling clock within the Wishbone slave (`wbgen2` feature).

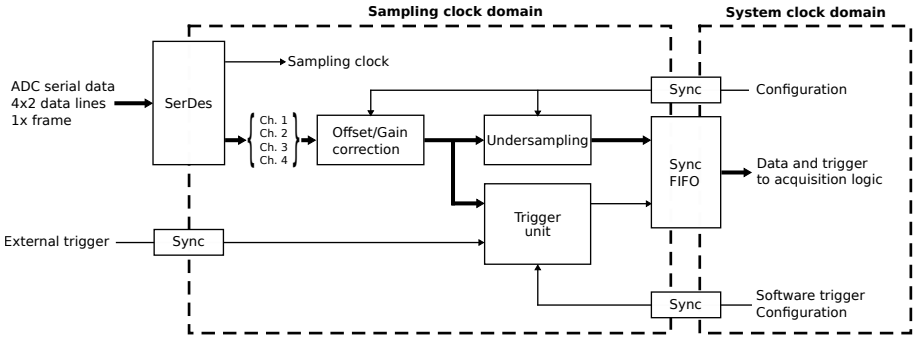


Figure 4.1: ADC core diagram (sampling clock domain).

The LTC2174 is by default configured as *2-Lane Output Mode, 16-Bit Serialization*. In the fmc-adc application, the default configuration is kept. The figure Figure 4.2 is an extract from the LTC2174 datasheet illustrating the *2-Lane Output Mode, 16-Bit Serialization* waveforms.

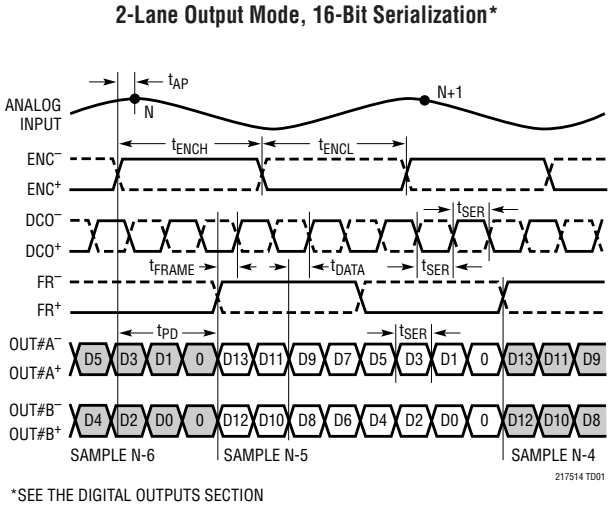


Figure 4.2: LTC2174 data output mode waveforms.

There is two 800Mb/s lanes per ADC channel. The eight data lanes and the frame rate (FR) lane are fed to a de-serialiser in the FPGA. The frame rate signal is used to align the de-serialiser to data words. The four channel data (16-bit) are concatenated together to form a 64-bit vector. As show in Figure 4.2, the two LSB bits of a data word are set to zero.

4.1 Control and Status Registers

Writing one to to the `FMC_CLK_OE` field of the ADC core control register enable the sampling clock (Si570 chip). Also, in order to use the input offset DACs, the `OFFSET_DAC_CLR_N` field must be set to one.

The field MAN_BITSLIP allows to 'manually' control the ADC data alignment in the de-serialiser. When TEST_DATA_EN is set, the ADC core writes the address pointer to the memory instead of the ADC samples. The fields TRIG_LED and ACQ_LED allows to control the FMC front panel LEDs. Those four fields are for test purpose only and must stay zero in normal operation.

When the sampling clock is enabled, the SERDES_PLL and SERDES_SYNCED field from the ADC core status register must be set to one.

4.2 Input Ranges

The Figure 4.3 shows a simplified schematics of the analogue input used for each channel. Each input can be independantly configured with one of the three available ranges; 100mV, 1V, 10V. Each range is defined as the maximum peak-to-peak input voltage. Independantly to the selected range, a 50ohms termination can be added to each input.

In addition to the three ranges for normal operation, there are three more configurations used for offset calibration of each range.

Opto-isolated analogue switches allow the different configurations. They are represented by normal switched in the simplified schematics.

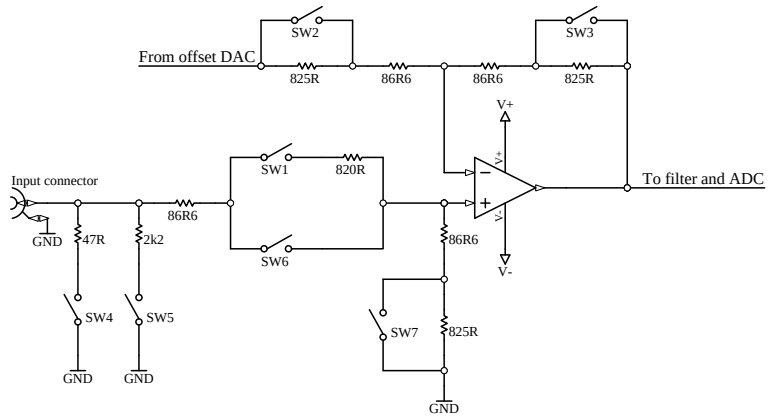


Figure 4.3: Simplified schematics of the analogue input.

Only the following input switch configurations are valid. For all others switch configurations, the behaviour is not defined and therefore shouldn't be used.

SW[7..1]	SW7	SW6	SW5	SW4	SW3	SW2	SW1	Description
0x23	OFF	ON	OFF	X	OFF	ON	ON	100mV range
0x11	OFF	OFF	ON	X	OFF	OFF	ON	1V range
0x45	ON	OFF	OFF	X	ON	OFF	ON	10V range
0x42	ON	OFF	OFF	X	OFF	ON	OFF	100mV range offset calibration
0x40	ON	OFF	OFF	X	OFF	OFF	OFF	1V range offset calibration
0x44	ON	OFF	OFF	X	ON	OFF	OFF	10V range offset calibration
0x00	X	OFF	OFF	OFF	X	X	OFF	Input disconnected
0x08	X	X	X	ON	X	X	X	50ohm termination

Table 4.1: Analogue input switches configurations.

4.3 Input Offset

Each channel has a 16-bit DAC allowing to apply a dc offset to the input signal. The voltage range of the DAC is 10V (-5V to +5V) and is independent from the selected input range. The following equation shows how to convert a digital value written to a DAC to an offset voltage.

```
v_dac = (v_ref * d_dac/0x8000) - v_ref
Where:
v_ref = DAC's voltage reference = 5V
d_dac = Digital value written to the DAC
v_dac = DAC voltage

Example:
0xFFFF =>  4.999V
0x8000 =>  0.000V
0x0000 => -5.000V
```

The following equation shows the relation between the input voltage and the offset (applied by the DAC). Note that the offset from the DAC is subtracted from the input voltage.

```
v_out = v_in - v_dac
Where:
v_in  = Input voltage
v_dac = DAC voltage
v_out = Output voltage (to filter and ADC)
```

4.4 Trigger

The trigger unit is made of a hardware and a software source. Each hardware and software sources can be enabled independantly. The two sources are then or'ed together to drive a delay generator. The delay generator allows to insert an defined number of sampling clock period before the trigger goes to the acquisition state machine.

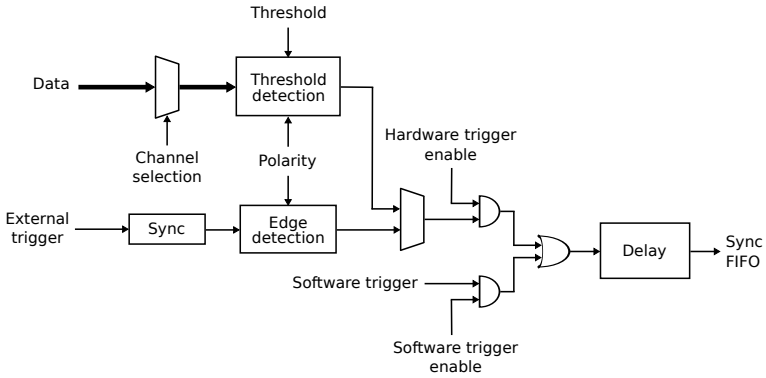


Figure 4.4: Trigger unit diagram.

The hardware trigger source can be either internal (based on an adc input channel) or external (dedicated trigger input). For both internal and external hardware triggers, the polarity can be selected between positive and negative slope (resp. rising and falling edge). By default the polatity is set to positive slope. The external trigger input is synchronised to the sampling clock. The external trigger pulse must be at least one sampling clock cycle wide. For the internal trigger source, the adc input channel and the threshold should be configured. By default, the channel 1 is selected and the threshold is set to 0. Note that the threshold is 16-bit signed (two's complement). The Figure 4.5 sketches the internal hardware trigger threshold behaviour. The software trigger source concists in a pulse generated when a write cycle is detected on the *Software trigger* register. The Figure 4.4 shows the different trigger configurations. For futher information on the trigger configuration registers see [ADC Core Registers], page 22.

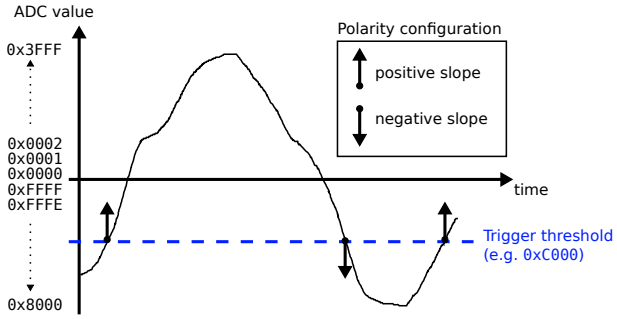


Figure 4.5: Internal hardware trigger trheshold.

4.5 Undersampling

The undersampling block is simply validating one in N samples and forwarding it to the acquisition logic. The number (N) is configured in the *Sample rate* register. If N > 1, then the trigger pulse is aligned to the next valid sample. If N = 1 all the samples are valid and therefore the trigger is always aligned. A value of N = 0 is treated as N = 1 in the firmware.

> **Note:** Undersampling might be unaccurately called decimation in the documentation or source code.

4.6 Time-taging

The time-tagging core contains two free running counters. The first one counts seconds and the second one counts system clock ticks (8ns resolution). The system clock ticks counter is alos called coarse counter. Those two counters can be set via the cores's Wishbone interface.

For example, the host computer can use the OS time to set the seconds counter and simply reset the coarse counter.

It is planned, in a later release, to set the time-tagging core counters using the White Rabbit core, for more details see Chapter 7 [Missing Features and Improvements], page 20.

5 Calibration

The calibration is done once during the production tests. It can be repeated afterwards with the production test suite (PTS) and the corresponding testbench. The calibration process gives the following four values per channel and per input range:

- ADC gain correction
- ADC offset correction
- DAC gain correction
- DAC offset correction

Note that the temperature during the calibration process is also measured. This could be used for later temperature compensated calibration value computing.

5.1 Calibration data storage

All the calibration values are stored in the FmcAdc100m14b4cha EEPROM. The EEPROM holds a sdbfs[1] file system. In addition to the calibration values, the EEPROM also contains mandatory IPMI[2] records specified in the FMC Standard VITA 57.1 (see table Table 5.1 for mapping).

Byte offset	File name	File Type	Description
0x0	ipmi.sdb	binary	IPMI records
0x100	calibration.sdb	binary	Calibration values
0x1000	.	binary	Directory vendor = 0xCE42 device = 0xC5BE045E

Table 5.1: EEPROM sdb file system.

Note that the vendor value 0xCE42 corresponds to CERN. While the device value 0xC5BE045E corresponds to the first 32-bit of the md5 sum of "fmc-adc-100m14b4cha".

5.2 Calibration Data Usage

5.2.1 ADC Calibration

Two registers per channel are implemented in the FPGA for ADC gain and offset correction. When an input range is selected, the corresponding gain/offset correction values must be loaded from the EEPROM to those registers.

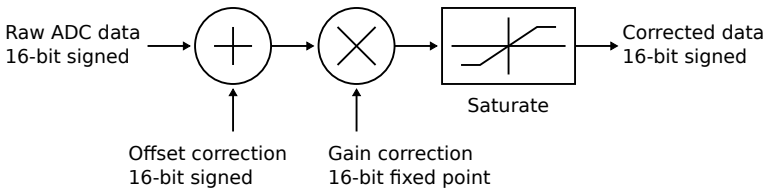


Figure 5.1: ADC offset and gain correction block.

The offset register takes a 16-bit signed value. The gain register takes a 16-bit fixed point value. The fixed point format is as follow:

[1] http://www.ohwr.org/attachments/download/1594/sdbfs-2012-09-19.pdf
[2] Platform Management FRU Information Storage Definition v1.0

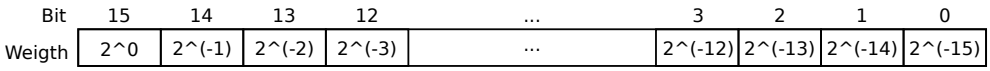


Figure 5.2: ADC gain register format.

Note: On FPGA start-up, the gain registers are set to 0x8000 (1.000) and the offset registers to 0x0000. This means a unit gain and no offset.

Note: After gain and offset correction, the two LSB of the data words can be different from zero.

Note: It is usually the driver's task to read the calibration data from the FMC EEPROM and load them to the corresponding registers. This has to be done once at start-up and then every time the input range is changed.

5.2.2 DAC Calibration

The DAC value is only set once before an acquisition. Therefore, there is no need to implement the gain and offset correction in the FPGA. The software controlling the fmc-adc must apply the DAC gain and offset correction prior to write a value to the DAC. As for the ADC correction values, there is one pair (offset, gain) of DAC correction values per input range.

Below is the formula to calculate the corrected DAC value (applying gain and offset correction):

```
c_val = ((val + offset) * gain/0x8000) - 0x8000
where:
c_val  = corrected value to write to DAC (16-bit unsigned)
val    = value from user (16-bit signed)
offset = DAC offset calibration value from EEPROM (16-bit signed)
gain   = DAC gain calibration value from EEPROM (16-bit fixed point)
```

6 Acquisition

This chapter describes the two modes of acquisition, single-shot and multi-shot. It also explains how the software is expected to control the fmc-adc acquisitions.

The Figure 6.1 shows the ADC core acquisition logic. The heart of the acquisition logic is a state machine driven by user commands (start, stop), the trigger signal and counters events (e.g. pre-trig done, etc...). The ADC samples are routed along a datapath (bold arrows), which depends on the acquisition mode. It is explained in detail in the Section 6.1 [Single-shot Mode], page 17 and Section 6.2 [Multi-shot Mode], page 18. The four channels data and the trigger are concatenated together and fed to a FIFO to be synchronised between the sampling clock domain and the system clock domain. Even if the LTC2174 ADC is 14-bit, the data of each channel is stored in a 16-bit word. Along the datapath, we call *sample* a 64-bit vector containing a sample for each channel. At the output of the ADC core, a flow control FIFO allows to cope with the memory controller temporary unavailabilities (due to DDR refresh cycles).

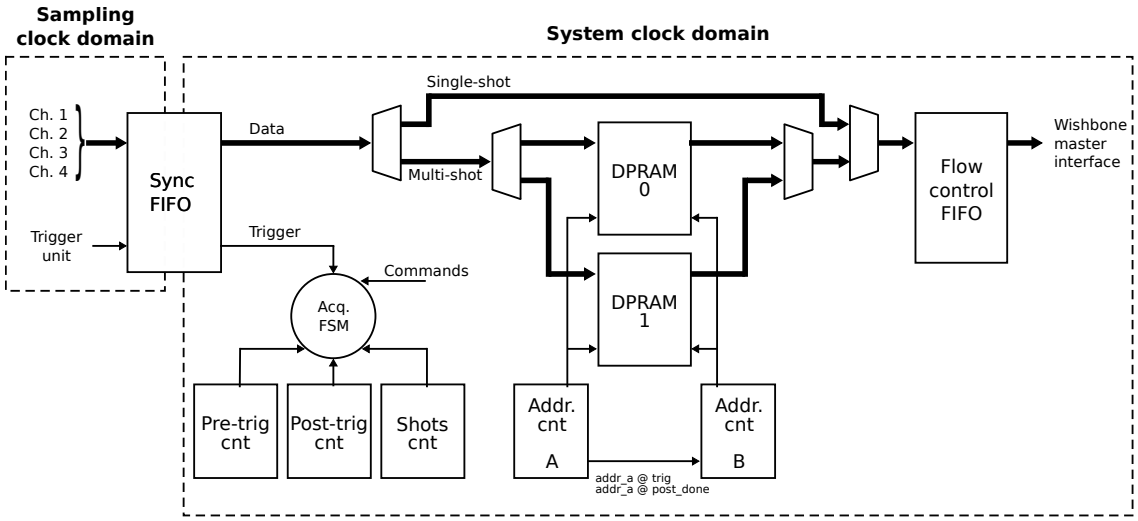


Figure 6.1: Acquisition logic diagram (system clock domain).

Samples are stored interleaved in the DDR memory. The Figure 6.2 illustrates the way samples are written, stored and read in the DDR memory. The DDR memory size is 2Gb or 256MB. It means that the maximum number of samples that can be stored is 128M samples ($2^{27} * 16$).

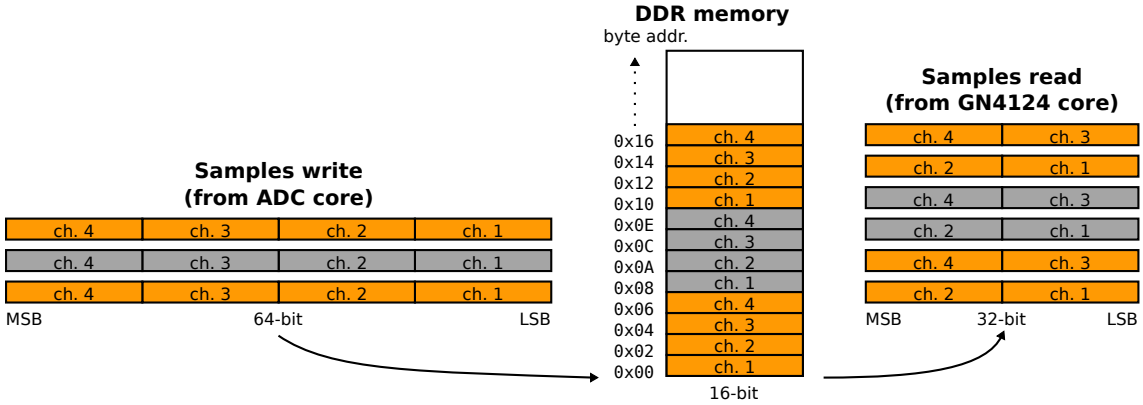


Figure 6.2: Illustation of samples storage in DDR memory.

The acquisition process is driven by a state machine. The Figure 6.3 represents its states and transitions. At start-up, the state machine is `IDLE`, waiting for an acquisistion start command (`ACQ_START`). Commands are sent to the state machine by writing in the `FSM_CMD` field of the

control register (see [ADC Core Registers], page 22). When a start command is received, the state machine goes to PRE_TRIG and stays in this state until the programmed number of pre-trigger samples are recorded. After that, it goes in WAIT_TRIG state and continue recording sample to memory. When a valid trigger is detected, the state machine moves to POST_TRIG. It will stays in this state until the programmed number of post-trigger samples is reached. Then, depending on the mode, the state machine either goes back to IDLE (single-shot mode) or to PRE_TRIG (multi-shot mode). When the acquisition is terminated (state machine back to IDLE) and all samples have been written to the DDR memory, only then the software can retrieve the samples using DMA transfer. An interrupt is generated when the acquisition ends.

Note: Start commands are taken into account only in IDLE state.

Note: Trigger are taken into account only in WAIT_TRIG state.

Note: A stop command will bring the state machine back to IDLE from any state.

Note: After a stop command, no end of acquisition interrupt is generated.

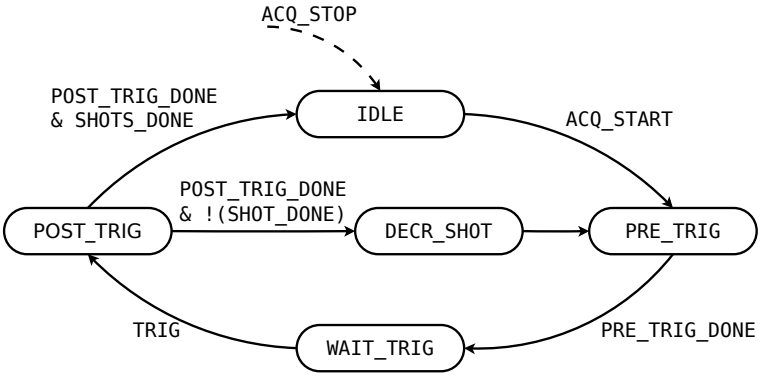


Figure 6.3: Acquisition state machine.

There are two LED on the fmc-adc front panel. The LED labeled ACQ is turned ON when the acquisition state machine is **not** in the IDLE state. The LED labeled TRIG flashes when a valid trigger is detected **and** the acquisition state machine is in the WAIT_TRIG state.

Note: In addition to the requested pre/post-trigger samples, an addition sample, corresponding to the trigger, will be recoded.

6.1 Single-shot Mode

The procedure below lists the different step of a single-shot acquisition process.

1. Configure acquisition (trigger, number of samples, interrupts, etc...).
2. Issue a start acquisition command (the acquisition state machine must be IDLE).
3. When a valid trigger is detected, an interrupt is generated.
4. At the end of the acquisition, another interrupt is generated.
5. Read trigger position register.
6. Configure the DMA to retreive data.
7. Start the DMA transfer (the acquisition state machine must be IDLE).
8. When the DMA transfer is done, an interrupt is generated.
9. The board is ready for a new acquisition start command.

In single-shot mode, the DDR memory is used as a circular buffer. When the acquisition starts, samples are direcly written to the DDR memory (via FIFOs). The acquisition logic stops writing to the memory when the configured number of pre/post-trigger samples is reached. It could happen that the write pointer reaches the top of the memory before the end of the

acquisition. In this case, the write pointer is reset to address zero and overwrite previous samples. In order to allow the software to retreive the requested samples (around the trigger), the *Trigger address* register s tores the write pointer address at the trigger moment.

> **Note:** The value stored in the *Trigger address* register is a sample address (64-bit word address).

> **Note:** Every new acquisition starts writing at address `0x0`.

The Figure 6.4 and Figure 6.5 illustrate the use of the DDR memory as a cicular buffer. The acquisition state machine is also represented.

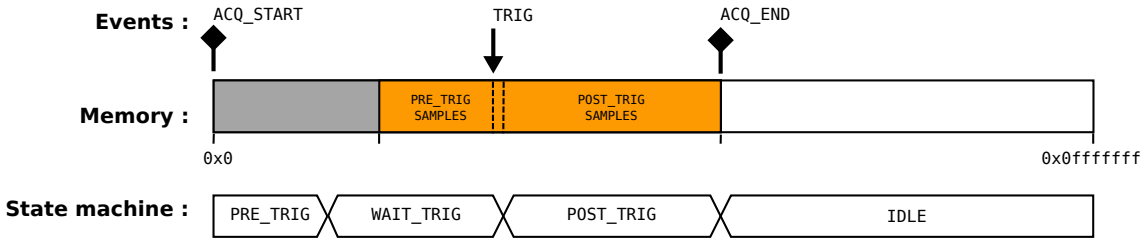


Figure 6.4: Single-shot mode acquisition example.

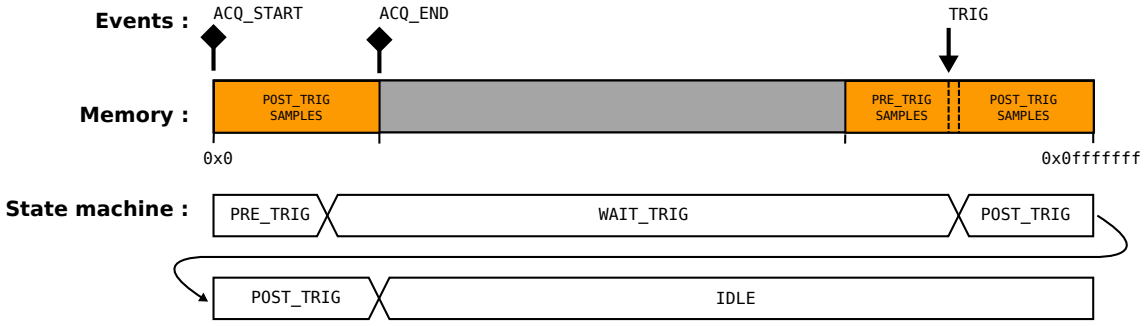


Figure 6.5: Single-shot mode acquisition example (overlapping DDR memory).

> **Note:** *Orange*: Samples written to memory and read back via DMA. *Grey*: Samples written to memory, but not read. *White*: Empty memory (or previous acquisition samples).

6.2 Multi-shot Mode

The multi-shot acquisition process is almost identical to the single-shot one, except that once the acquisition is started it will go around the state machine as many time as the number of configured shots. It means that if the board is configured for N shots, it will generate N trigger interrupts and then another interrupt at the end of the acquisition.

Unlike the single-mode acquisition, in multi-shot, the DDR memory is not used as a circular buffer. Instead, two dual port RAM (dpram) are implemented inside the FPGA. Those dprams are alternatively used as circular buffer for each shot. Even shots uses dpram0 and odd shots dpram1.

> **Note:** The dprams are 2048 samples deep. It means that the total number of samples (pre-trigger + post-trigger) for a shot cannot exceed 2048.

When a shot is finished, the correcponding dpram samples are written to the DDR memory. Only the pre-trigger and post-trigger samples are written. The first shot is written starting at address `0x0`. Then the second shot is written right after the last post-trigger sample of the first shot. The Figure 6.6 shows the shots organisation in the DDR memory.

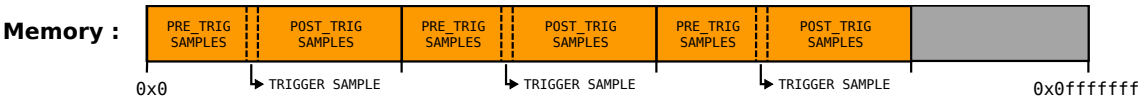


Figure 6.6: DDR memory usage in multi-shot mode acquisition.

Note: The number of samples per shot stored in memory is equal to: number of pre-trigger samples + number of post-trigger samples + 1 (trigger sample).

7 Missing Features and Improvements

7.1 To be done before next release

- Remove huge files from git repo. **!!! This will change all commits sha !!!**
- Add a reference section (bibliography).

7.2 For a later release

- Remove carrier SPI master from mapping -> shift other slaves base addresses.
- Add WR core; 1)for time-tags, 2)for sampling clock control
  - Define behaviour when WR is desconnected.
  - Assign signals to SPEC front panel LEDs.
- Add Etherbone support.
- Remove mutli-irq register from interrupt controller.
  Perhaps add a counter per interrupt source instead.
- Remove unused 250MHz clock signals and buffer.
- Unify address inferfaces: put all in bytes (wishbone addr, trig pointer, ...)
  - Change GN4142-core WB bus(es) to byte address.
  - Change DDR-core WB bus(es) to byte address?
- Add error flags (interrupt?):
  - Instead of overwriting memory for a given acquisition.
  - If read during acquisition (or even block read during acq?).
- Rename decimation (and "sample rate" register) in under-sampling.
- Use 200MHz clock for WB bus from ddr-ctrl to gn4124-core.
- Clean-up adc core WB interface to DDR -> use only one clock (=> sys_clk).
- Replace all Xilinx FIFO by generic ones from general-cores lib (! last time I tried, it broke the DMA.).
- Test sampling clocks from 10MHz to 105MHz.
- Add sampling clock presence flag. Or better a sampling clock frequency register.
- Add over-heat and input over-load interrupts? (from original specification)
- Time-tag for every trigger in multi-shot. -> trigger time-tag array
- Review reset logic.
- Generate an end of acquisition interrupt after an acquisition stop command?
- Remove meta-info field in time-tags?
- Move sdb device descriptions from top to the wishbone_pkg.vhd (general-cores lib).
- Make the project ucfgen friendly.
  - Put all mezzanine related cores in a wrapper (fmc adc block).
  - Add a crossbar inside the fmc adc block -> check impact on sdb.
- Include the git tree in a .tar.gz along with the .bin file (in the files section) for each release.
  -> modify the Release chapter accordingly.

Appendix A

A.1 Calibration Data Storage in EEPROM

Tables Table A.1 and Table A.2 shows the calibration data types and the arrangement in the binary file. The first column "Byte offset" represents the offset within the binary file.

Byte offset	Input range	Description	Type
0x0	10V	Offset correction channel 1	16-bit signed
0x2		Offset correction channel 2	16-bit signed
0x4		Offset correction channel 3	16-bit signed
0x6		Offset correction channel 4	16-bit signed
0x8		Gain correction channel 1	16-bit unsigned
0xA		Gain correction channel 2	16-bit unsigned
0xC		Gain correction channel 3	16-bit unsigned
0xE		Gain correction channel 4	16-bit unsigned
0x10		Temperature	16-bit unsigned * 0.01°
0x12	1V	Offset correction channel 1	16-bit signed
0x14		Offset correction channel 2	16-bit signed
0x16		Offset correction channel 3	16-bit signed
0x18		Offset correction channel 4	16-bit signed
0x1A		Gain correction channel 1	16-bit unsigned
0x1C		Gain correction channel 2	16-bit unsigned
0x1E		Gain correction channel 3	16-bit unsigned
0x20		Gain correction channel 4	16-bit unsigned
0x22		Temperature	16-bit unsigned * 0.01°
0x24	100mV	Offset correction channel 1	16-bit signed
0x26		Offset correction channel 2	16-bit signed
0x28		Offset correction channel 3	16-bit signed
0x2A		Offset correction channel 4	16-bit signed
0x2C		Gain correction channel 1	16-bit unsigned
0x2E		Gain correction channel 2	16-bit unsigned
0x30		Gain correction channel 3	16-bit unsigned
0x32		Gain correction channel 4	16-bit unsigned
0x34		Temperature	16-bit unsigned * 0.01°

Table A.1: ADC calibration data stored in EEPROM (calibration.sdb file).

Byte offset	Input range	Description	Type
0x36	10V	Offset correction channel 1	16-bit signed
0x38		Offset correction channel 2	16-bit signed
0x3A		Offset correction channel 3	16-bit signed
0x3C		Offset correction channel 4	16-bit signed
0x3E		Gain correction channel 1	16-bit unsigned
0x40		Gain correction channel 2	16-bit unsigned
0x42		Gain correction channel 3	16-bit unsigned
0x44		Gain correction channel 4	16-bit unsigned
0x46		Temperature	16-bit unsigned * 0.01°
0x48	1V	Offset correction channel 1	16-bit signed
0x4A		Offset correction channel 2	16-bit signed
0x4C		Offset correction channel 3	16-bit signed
0x4E		Offset correction channel 4	16-bit signed
0x50		Gain correction channel 1	16-bit unsigned
0x52		Gain correction channel 2	16-bit unsigned
0x54		Gain correction channel 3	16-bit unsigned
0x56		Gain correction channel 4	16-bit unsigned
0x58		Temperature	16-bit unsigned * 0.01°
0x5A	100mV	Offset correction channel 1	16-bit signed
0x5C		Offset correction channel 2	16-bit signed
0x5E		Offset correction channel 3	16-bit signed
0x60		Offset correction channel 4	16-bit signed
0x62		Gain correction channel 1	16-bit unsigned
0x64		Gain correction channel 2	16-bit unsigned
0x66		Gain correction channel 3	16-bit unsigned
0x68		Gain correction channel 4	16-bit unsigned
0x6A		Temperature	16-bit unsigned * 0.01°

Table A.2: DAC calibration data stored in EEPROM (calibration.sdb file).

Appendix B ADC Core Registers

B.1 Memory map summary

Address	Type	Prefix	Name
0x0	REG	ctl	Control register
0x4	REG	sta	Status register
0x8	REG	trig_cfg	Trigger configuration
0xc	REG	trig_dly	Trigger delay
0x10	REG	sw_trig	Software trigger
0x14	REG	shots	Number of shots
0x18	REG	trig_pos	Trigger address register
0x1c	REG	sr	Sample rate
0x20	REG	pre_samples	Pre-trigger samples

0x24	REG	post_samples	Post-trigger samples
0x28	REG	samples_cnt	Samples counter
0x2c	REG	ch1_ctl	Channel 1 control register
0x30	REG	ch1_sta	Channel 1 status register
0x34	REG	ch1_gain	Channel 1 gain calibration register
0x38	REG	ch1_offset	Channel 1 offset calibration register
0x3c	REG	ch2_ctl	Channel 2 control register
0x40	REG	ch2_sta	Channel 2 status register
0x44	REG	ch2_gain	Channel 2 gain calibration register
0x48	REG	ch2_offset	Channel 2 offset calibration register
0x4c	REG	ch3_ctl	Channel 3 control register
0x50	REG	ch3_sta	Channel 3 status register
0x54	REG	ch3_gain	Channel 3 gain calibration register
0x58	REG	ch3_offset	Channel 3 offset calibration register
0x5c	REG	ch4_ctl	Channel 4 control register
0x60	REG	ch4_sta	Channel 4 status register
0x64	REG	ch4_gain	Channel 4 gain calibration register
0x68	REG	ch4_offset	Channel 4 offset calibration register

B.2 `ctl` - Control register

Bits	Access	Prefix	Default	Name
1...0	R/W	FSM_CMD	X	State machine commands (ignore on read)
2	R/W	FMC_CLK_OE	X	FMC Si750 output enable
3	R/W	OFFSET_DAC_CLR_N	X	Offset DACs clear (active low)
4	W/O	MAN_BITSLIP	X	Manual serdes bitslip (ignore on read)
5	R/W	TEST_DATA_EN	X	Enable test data
6	R/W	TRIG_LED	X	Manual TRIG LED
7	R/W	ACQ_LED	X	Manual ACQ LED
31...8	R/W	RESERVED	X	Reserved

Field	Description
fsm_cmd	1: ACQ_START (start acquisition, only when FSM is idle) 2: ACQ_STOP (stop acquisition, anytime)
test_data_en	Write the address counter value instead of ADC data to DDR
trig_led	Manual control of the front panel TRIG LED
acq_led	Manual control of the front panel ACQ LED
reserved	Ignore on read, write with 0's

B.3 sta - Status register

Bits	Access	Prefix	Default	Name
2...0	R/O	FSM	X	State machine status
3	R/O	SERDES_PLL	X	SerDes PLL status
4	R/O	SERDES_SYNCED	X	SerDes synchronization status
31...5	R/O	RESERVED	X	Reserved

Field	Description
fsm	States: 0: illegal 1: IDLE 2: PRE_TRIG 3: WAIT_TRIG 4: POST_TRIG 5: DECR_SHOT 6: illegal 7: illegal
serdes_pll	Sampling clock recovery PLL. 0: not locked 1: locked
serdes_synced	0: bitslip in progress 1: serdes synchronized
reserved	Ignore on read, write with 0's

B.4 trig_cfg - Trigger configuration

Bits	Access	Prefix	Default	Name
0	R/W	HW_TRIG_SEL	X	Hardware trigger selection
1	R/W	HW_TRIG_POL	X	Hardware trigger polarity
2	R/W	HW_TRIG_EN	X	Hardware trigger enable
3	R/W	SW_TRIG_EN	X	Software trigger enable
5...4	R/W	INT_TRIG_SEL	X	Channel selection for internal trigger
15...6	R/W	RESERVED	X	Reserved
31...16	R/W	INT_TRIG_THRES	X	Threshold for internal trigger

Field	Description
hw_trig_sel	0: internal (data threshold) 1: external (front panel trigger input)
hw_trig_pol	0: positive edge/slope 1: negative edge/slope

hw_trig_en	0: disable 1: enable
sw_trig_en	0: disable 1: enable
int_trig_sel	00: channel 1 01: channel 2 10: channel 3 11: channel 4
reserved	Ignore on read, write with 0's
int_trig_thres	Treated as binary two's complement and compared to raw ADC data

B.5 trig_dly - Trigger delay

Bits	Access	Prefix	Default	Name
31...0	R/W	TRIG_DLY	X	Trigger delay value

Field	Description
trig_dly	Delay to apply on the trigger in sampling clock period. The default clock frequency is 100MHz (period = 10ns).

B.6 sw_trig - Software trigger

Writing (anything) to this register generates a software trigger.

Bits	Access	Prefix	Default	Name
31...0	W/O	SW_TRIG	X	Software trigger (ignore on read)

Field	Description

B.7 shots - Number of shots

Bits	Access	Prefix	Default	Name
15...0	R/W	NB	X	Number of shots
31...16	R/W	RESERVED	X	Reserved

Field	Description
nb	Number of shots required in multi-shot mode, set to one for single-shot mode.
reserved	Ignore on read, write with 0's

B.8 trig_pos - Trigger address register

Bits	Access	Prefix	Default	Name
31...0	R/O	TRIG_POS	X	Trigger address

Field	Description
trig_pos	Trigger address in DDR memory. Only used in single-shot mode.

B.9 sr - Sample rate

Bits	Access	Prefix	Default	Name
15...0	R/W	DECI	X	Sample rate decimation
31...16	R/W	RESERVED	X	Reserved

Field	Description
deci	Decimation factor. Takes one sample every N samples and discards the others (N = decimation factor).
reserved	Ignore on read, write with 0's

B.10 pre_samples - Pre-trigger samples

Bits	Access	Prefix	Default	Name
31...0	R/W	PRE_ SAMPLES	X	Pre-trigger samples

Field	Description
pre_samples	Number of requested pre-trigger samples

B.11 post_samples - Post-trigger samples

Bits	Access	Prefix	Default	Name
31...0	R/W	POST_ SAMPLES	X	Post-trigger samples

Field	Description
post_ samples	Number of requested post-trigger samples

B.12 samples_cnt - Samples counter

Bits	Access	Prefix	Default	Name
31...0	R/O	SAMPLES_ CNT	X	Samples counter

Field	Description
samples_cnt	Counts the number of samples. It is reset on START and then counts the number of pre-trigger + post-trigger samples

B.13 ch1_ctl - Channel 1 control register

Bits	Access	Prefix	Default	Name
6...0	R/W	SSR	X	Solid state relays control for channel 1
31...7	R/W	RESERVED	X	Reserved

Field	Description

ssr	Controls input voltage range, termination and DC offset error calibration 0x23: 100mV range 0x11: 1V range 0x45: 10V range 0x00: Open input 0x42: 100mV range calibration 0x40: 1V range calibration 0x44: 10V range calibration Bit3 is indepandant of the others and enables the 50ohms termination.
reserved	Ignore on read, write with 0's

B.14 ch1_sta - Channel 1 status register

Bits	Access	Prefix	Default	Name
15...0	R/O	VAL	X	Channel 1 current ADC value
31...16	R/O	RESERVED	X	Reserved

Field	Description
val	Current ADC raw value. The format is binary two's complement.
reserved	Ignore on read, write with 0's

B.15 ch1_gain - Channel 1 gain calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Gain calibration for channel 1
31...16	R/W	RESERVED	X	Reserved

Field	Description
val	Gain applied to all data coming from the ADC. Fixed point format: Bit 15 = 2^0, bit 14 = 2^(-1), bit 13 = 2^(-2), ... , bit 1 = 2^(-14), bit 0 = 2^(-15)
reserved	Ignore on read, write with 0's

B.16 ch1_offset - Channel 1 offset calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Offset calibration for channel 1
31...16	R/W	RESERVED	X	Reserved

Field	Description
val	Offset applied to all data coming from the ADC. The format is binary two's complement.
reserved	Ignore on read, write with 0's

B.17 ch2_ctl - Channel 2 control register

Bits	Access	Prefix	Default	Name
6...0	R/W	SSR	X	Solid state relays control for channel 2

31...7	R/W	RESERVED	X	Reserved

Field	Description
ssr	Controls input voltage range, termination and DC offset error calibration 0x23: 100mV range 0x11: 1V range 0x45: 10V range 0x00: Open input 0x42: 100mV range calibration 0x40: 1V range calibration 0x44: 10V range calibration Bit3 is indepandant of the others and enables the 50ohms termination.
reserved	Ignore on read, write with 0's

B.18 ch2_sta - Channel 2 status register

Bits	Access	Prefix	Default	Name
15...0	R/O	VAL	X	Channel 2 current ACD value
31...16	R/O	RESERVED	X	Reserved

Field	Description
val	Current ADC raw value. The format is binary two's complement.
reserved	Ignore on read, write with 0's

B.19 ch2_gain - Channel 2 gain calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Gain calibration for channel 2
31...16	R/W	RESERVED	X	Reserved

Field	Description
val	Gain applied to all data coming from the ADC. Fixed point format: Bit 15 = 2^0, bit 14 = 2^(-1), bit 13 = 2^(-2), ... , bit 1 = 2^(-14), bit 0 = 2^(-15)
reserved	Ignore on read, write with 0's

B.20 ch2_offset - Channel 2 offset calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Offset calibration for channel 2
31...16	R/W	RESERVED	X	Reserved

Field	Description
val	Offset applied to all data coming from the ADC. The format is binary two's complement.
reserved	Ignore on read, write with 0's

B.21 ch3_ctl - Channel 3 control register

Bits	Access	Prefix	Default	Name
6...0	R/W	SSR	X	Solid state relays control for channel 3
31...7	R/W	RESERVED	X	Reserved

Field	Description
ssr	Controls input voltage range, termination and DC offset error calibration 0x23: 100mV range 0x11: 1V range 0x45: 10V range 0x00: Open input 0x42: 100mV range calibration 0x40: 1V range calibration 0x44: 10V range calibration Bit3 is indepandant of the others and enables the 50ohms termination.
reserved	Ignore on read, write with 0's

B.22 ch3_sta - Channel 3 status register

Bits	Access	Prefix	Default	Name
15...0	R/O	VAL	X	Channel 3 current ADC value
31...16	R/O	RESERVED	X	Reserved

Field	Description
val	Current ADC raw value. The format is binary two's complement.
reserved	Ignore on read, write with 0's

B.23 ch3_gain - Channel 3 gain calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Gain calibration for channel 3
31...16	R/W	RESERVED	X	Reserved

Field	Description
val	Gain applied to all data coming from the ADC. Fixed point format: Bit 15 = 2^0, bit 14 = 2^(-1), bit 13 = 2^(-2), ... , bit 1 = 2^(-14), bit 0 = 2^(-15)
reserved	Ignore on read, write with 0's

B.24 ch3_offset - Channel 3 offset calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Offset calibration for channel 3
31...16	R/W	RESERVED	X	Reserved

Field	Description

val	Offset applied to all data coming from the ADC. The format is binary two's complement.
reserved	Ignore on read, write with 0's

B.25 ch4_ctl - Channel 4 control register

Bits	Access	Prefix	Default	Name
6...0	R/W	SSR	X	Solid state relays control for channel 4
31...7	R/W	RESERVED	X	Reserved

Field	Description
ssr	Controls input voltage range, termination and DC offset error calibration 0x23: 100mV range 0x11: 1V range 0x45: 10V range 0x00: Open input 0x42: 100mV range calibration 0x40: 1V range calibration 0x44: 10V range calibration Bit3 is indepandant of the others and enables the 50ohms termination.
reserved	Ignore on read, write with 0's

B.26 ch4_sta - Channel 4 status register

Bits	Access	Prefix	Default	Name
15...0	R/O	VAL	X	Channel 4 current ADC value
31...16	R/O	RESERVED	X	Reserved

Field	Description
val	Current ADC raw value. The format is binary two's complement.
reserved	Ignore on read, write with 0's

B.27 ch4_gain - Channel 4 gain calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Gain calibration for channel 4
31...16	R/W	RESERVED	X	Reserved

Field	Description
val	Gain applied to all data coming from the ADC. Fixed point format: Bit 15 = 2^0, bit 14 = 2^(-1), bit 13 = 2^(-2), ... , bit 1 = 2^(-14), bit 0 = 2^(-15)
reserved	Ignore on read, write with 0's

B.28 ch4_offset - Channel 4 offset calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	X	Offset calibration for channel 4

31...16	R/W	RESERVED	X	Reserved

Field	Description
val	Offset applied to all data coming from the ADC. The format is binary two's complement.
reserved	Ignore on read, write with 0's

Appendix C Interrupt Controller Registers

C.1 Memory map summary

Address	Type	Prefix	Name
0x0	REG	multi_irq	Multiple interrupt register
0x4	REG	src	Interrupt sources register
0x8	REG	en_mask	Interrupt enable mask register

C.2 multi_irq - Multiple interrupt register

Multiple interrupts occurs before irq source is read. Write '1' to clear a bit.

Bit 0: DMA done. Bit 1: DMA error. Bit 2: Trigger. Bit 3: Acquisition end.

Bits	Access	Prefix	Default	Name
31...0	R/W	MULTI_IRQ	X	Multiple interrupt

Field	Description

C.3 src - Interrupt sources register

Indicates the interrupt source. Write '1' to clear a bit.

Bit 0: DMA done. Bit 1: DMA error. Bit 2: Trigger. Bit 3: Acquisition end.

Bits	Access	Prefix	Default	Name
31...0	R/W	SRC	X	Interrupt sources

Field	Description

C.4 en_mask - Interrupt enable mask register

Bit mask to independently enable interrupt sources.

Bit 0: DMA done. Bit 1: DMA error. Bit 2: Trigger. Bit 3: Acquisition end.

Bits	Access	Prefix	Default	Name
31...0	R/W	EN_MASK	X	Interrupt enable mask

Field	Description

Appendix D Time-tagging Core Registers

D.1 Memory map summary

Address	Type	Prefix	Name
0x0	REG	seconds	Timetag seconds register
0x4	REG	coarse	Timetag coarse time register, system clock ticks (125MHz)
0x8	REG	trig_tag_meta	Trigger time-tag metadata register
0xc	REG	trig_tag_seconds	Trigger time-tag seconds register
0x10	REG	trig_tag_coarse	Trigger time-tag coarse time (system clock ticks 125MHz) register
0x14	REG	trig_tag_fine	Trigger time-tag fine time register, always 0 (used for time-tag format compatibility)
0x18	REG	acq_start_tag_meta	Acquisition start time-tag metadata register
0x1c	REG	acq_start_tag_seconds	Acquisition start time-tag seconds register
0x20	REG	acq_start_tag_coarse	Acquisition start time-tag coarse time (system clock ticks 125MHz) register
0x24	REG	acq_start_tag_fine	Acquisition start time-tag fine time register, always 0 (used for time-tag format compatibility)
0x28	REG	acq_stop_tag_meta	Acquisition stop time-tag metadata register
0x2c	REG	acq_stop_tag_seconds	Acquisition stop time-tag seconds register
0x30	REG	acq_stop_tag_coarse	Acquisition stop time-tag coarse time (system clock ticks 125MHz) register
0x34	REG	acq_stop_tag_fine	Acquisition stop time-tag fine time register, always 0 (used for time-tag format compatibility)
0x38	REG	acq_end_tag_meta	Acquisition end time-tag metadata register
0x3c	REG	acq_end_tag_seconds	Acquisition end time-tag seconds register
0x40	REG	acq_end_tag_coarse	Acquisition end time-tag coarse time (system clock ticks 125MHz) register

0x44	REG	acq_end_tag_fine	Acquisition end time-tag fine time register, always 0 (used for time-tag format compatibility)

D.2 seconds - Timetag seconds register

Seconds counter. Incremented everytime the coarse counter overflows.

Bits	Access	Prefix	Default	Name
31...0	R/W	SECONDS	X	Timetag seconds

Field	Description

D.3 coarse - Timetag coarse time register, system clock ticks (125MHz)

Coarse time counter clocked by 125MHz system clock. Counts from 0 to 125000000.

Bits	Access	Prefix	Default	Name
31...0	R/W	COARSE	X	Timetag coarse time

Field	Description

D.4 trig_tag_meta - Trigger time-tag metadata register

Bits	Access	Prefix	Default	Name
31...0	R/O	TRIG_TAG_META	X	Trigger time-tag metadata

Field	Description
trig_tag_meta	Holds time-tag metadata of the last trigger event

D.5 trig_tag_seconds - Trigger time-tag seconds register

Bits	Access	Prefix	Default	Name
31...0	R/O	TRIG_TAG_SECONDS	X	Trigger time-tag seconds

Field	Description
trig_tag_seconds	Holds time-tag seconds of the last trigger event

D.6 trig_tag_coarse - Trigger time-tag coarse time (system clock ticks 125MHz) register

Bits	Access	Prefix	Default	Name
31...0	R/O	TRIG_TAG_COARSE	X	Trigger time-tag coarse time

Field	Description
trig_tag_coarse	Holds time-tag coarse time of the last trigger event

D.7 trig_tag_fine - Trigger time-tag fine time register, always 0 (used for time-tag format compatibility)

Bits	Access	Prefix	Default	Name
31...0	R/O	TRIG_TAG_FINE	X	Trigger time-tag fine time

Field	Description
trig_tag_fine	Holds time-tag fine time of the last trigger event

D.8 acq_start_tag_meta - Acquisition start time-tag metadata register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_START_TAG_META	X	Acquisition start time-tag metadata

Field	Description
acq_start_tag_meta	Holds time-tag metadata of the last acquisition start event

D.9 acq_start_tag_seconds - Acquisition start time-tag seconds register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_START_TAG_SECONDS	X	Acquisition start time-tag seconds

Field	Description
acq_start_tag_seconds	Holds time-tag seconds of the last acquisition start event

D.10 acq_start_tag_coarse - Acquisition start time-tag coarse time (system clock ticks 125MHz) register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_START_TAG_COARSE	X	Acquisition start time-tag coarse time

Field	Description
acq_start_tag_coarse	Holds time-tag coarse time of the last acquisition start event

D.11 acq_start_tag_fine - Acquisition start time-tag fine time register, always 0 (used for time-tag format compatibility)

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_START_TAG_FINE	X	Acquisition start time-tag fine time

Field	Description
acq_start_tag_fine	Holds time-tag fine time of the last acquisition start event

D.12 acq_stop_tag_meta - Acquisition stop time-tag metadata register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_STOP_TAG_META	X	Acquisition stop time-tag metadata

Field	Description
acq_stop_tag_meta	Holds time-tag metadata of the last acquisition stop event

D.13 acq_stop_tag_seconds - Acquisition stop time-tag seconds register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_STOP_TAG_SECONDS	X	Acquisition stop time-tag seconds

Field	Description
acq_stop_tag_seconds	Holds time-tag seconds of the last acquisition stop event

D.14 acq_stop_tag_coarse - Acquisition stop time-tag coarse time (system clock ticks 125MHz) register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_STOP_TAG_COARSE	X	Acquisition stop time-tag coarse time

Field	Description
acq_stop_tag_coarse	Holds time-tag coarse time of the last acquisition stop event

D.15 acq_stop_tag_fine - Acquisition stop time-tag fine time register, always 0 (used for time-tag format compatibility)

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_STOP_TAG_FINE	X	Acquisition stop time-tag fine time

Field	Description
acq_stop_tag_fine	Holds time-tag fine time of the last acquisition stop event

D.16 acq_end_tag_meta - Acquisition end time-tag metadata register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_END_TAG_META	X	Acquisition end time-tag metadata

Field	Description
acq_end_tag_meta	Holds time-tag metadata of the last acquisition end event

D.17 acq_end_tag_seconds - Acquisition end time-tag seconds register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_END_TAG_SECONDS	X	Acquisition end time-tag seconds

Field	Description
acq_end_tag_seconds	Holds time-tag seconds of the last acquisition end event

D.18 acq_end_tag_coarse - Acquisition end time-tag coarse time (system clock ticks 125MHz) register

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_END_TAG_COARSE	X	Acquisition end time-tag coarse time

Field	Description

acq_end_ tag_coarse	Holds time-tag coarse time of the last acquisition end event

D.19 acq_end_tag_fine - Acquisition end time-tag fine time register, always 0 (used for time-tag format compatibility)

Bits	Access	Prefix	Default	Name
31...0	R/O	ACQ_END_ TAG_FINE	X	Acquisition end time-tag fine time

Field	Description
acq_end_ tag_fine	Holds time-tag fine time of the last acquisition end event

Appendix E Carrier Registers

E.1 Memory map summary

Address	Type	Prefix	Name
0x0	REG	carrier	Carrier type and PCB version
0x4	REG	stat	Status
0x8	REG	ctrl	Control

E.2 carrier - Carrier type and PCB version

Bits	Access	Prefix	Default	Name
3...0	R/O	PCB_REV	X	PCB revision
15...4	R/O	RESERVED	X	Reserved register
31...16	R/O	TYPE	X	Carrier type

Field	Description
pcb_rev	Binary coded PCB layout revision.
reserved	Ignore on read, write with 0's.
type	Carrier type identifier 1 = SPEC 2 = SVEC 3 = VFC 4 = SPEXI

E.3 stat - Status

Bits	Access	Prefix	Default	Name
0	R/O	FMC_PRES	X	FMC presence
1	R/O	P2L_PLL_ LCK	X	GN4142 core P2L PLL status
2	R/O	SYS_PLL_ LCK	X	System clock PLL status

3	R/O	DDR3_CAL_DONE	X	DDR3 calibration status
31...4	R/O	RESERVED	X	Reserved

Field	Description
fmc_pres	0: FMC slot is populated 1: FMC slot is not populated.
p2l_pll_lck	0: not locked 1: locked.
sys_pll_lck	0: not locked 1: locked.
ddr3_cal_done	0: not done 1: done.
reserved	Ignore on read, write with 0's.

E.4 ctrl - Control

Bits	Access	Prefix	Default	Name
0	R/W	LED_GREEN	X	Green LED
1	R/W	LED_RED	X	Red LED
2	R/W	DAC_CLR_N	X	DAC clear
31...3	R/W	RESERVED	X	Reserved

Field	Description
led_green	Manual control of the front panel green LED (unused in the fmc-adc application)
led_red	Manual control of the front panel red LED (unused in the fmc-adc application)
dac_clr_n	Active low clear signal for VCXO DACs
reserved	Ignore on read, write with 0's

Appendix F References

F.1 References

Appendix G Glossary

G.1 Glossary

Local bus The **local bus** is the interface between the GN4124 and the FPGA.

Pulse In this document, a **pulse** refers to a one clock tick wide pulse.

Tick A clock **tick** corresponds to a period of the clock.