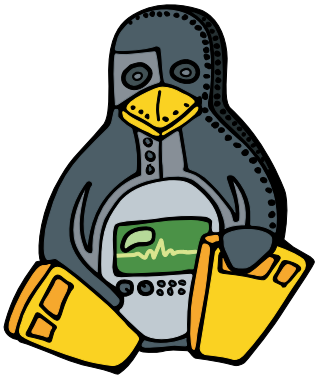
FmcAdc100m14b4cha Gateware Guide

April 2016 - Release 4.1
For PCIe (SPEC) and VME64x (SVEC) FMC Carriers



Matthieu Cattin (CERN)
Dimitrios Lampridis (CERN)

Table of Contents

Introduction

This document describes the gateware developed to support the FmcAdc100m14b4cha (later refered to as fmc-adc) mezzanine card on the SPEC[1] and SVEC[2] 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-adc 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, nor details about it's hardware design.

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-gw`

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

`Documents`[3]
: contains the `.bin` FPGA binary files and the `.pdf` documentation 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` (or `svec-fmc-adc-v#maj.#min`) where `#maj` represent the major release version of the gateware and `#min` the minor one (e.g `spec-fmc-adc-v1.2`). The released FPGA binary files follow the same naming convention.

The git commit hash has to be written in the sdb meta-information, therefore a release consists of two commits. The commit coming right after the tagged one contains the updated sdb meta-information file, the ise project and the synthesis, place&route, timing, and the reports.

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 gateware, for a given major version, should be backwards compatible. If the interface with the driver changes, the major version should be incremented. This means that driver versions 1.x should work with any gateware version 1.x. But the driver version 2.0 might not work with the gateware version 1.1.

[1] `http://www.ohwr.org/projects/spec`
[2] `http://www.ohwr.org/projects/svec`
[3] `http://www.ohwr.org/projects/fmc-adc-100m14b4cha-gw/documents`
[4] `http://www.ohwr.org/projects/fmc-adc-100m14b4cha-gw/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` (version 2.1).
2. Get fmc-adc hdl sources.
   `git clone git://ohwr.org/fmc-projects/fmc-adc-100m14b4cha-gw.git <src_dir>`
3. Goto the synthesis directory.
   `cd <src_dir>/hdl/<carrier>/syn/`
4. Fetch the dependencies and generate a synthesis Makefile.
   `hdlmake`
5. Perform synthesis, place, route and generate FPGA bitstream.
   `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/ip_cores/`
: Location of fetched and generated hdl cores and libraries.

`hdl/<carrier>/rtl/`
: Carrier related hdl sources.

`hdl/<carrier>/wb_gen/`
: Carrier related `wbgen2` sources, html documentation and C header file.

`hdl/<carrier>/syn/`
: Synthesis directory for selected 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/<carrier>/sim/`
`hdl/<carrier>/testbench/`
: Carrier related simulation files and testbenches.

`hdl/<carrier>/chipscope/`
: 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 gateware release. In order to identify the source files used in a given release, refer to the `Manifest.py` files.

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

2.3 Dependencies

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

general-cores

```
repo : git://ohwr.org/hdl-core-lib/general-cores.git
commit: c26ee857158e4a65fd9d2add8b63fcb6fb4691ea
```

ddr3-sp6-core

```
repo : git://ohwr.org/hdl-core-lib/ddr3-sp6-core.git
commit: e4d6755cc9c9c5cb005ce12eb82b12552922b882
```

gn4124-core (spec carrier only)

```
repo : git://ohwr.org/hdl-core-lib/gn4124-core.git
commit: e0dcb3f9a3e6804f64c544743bdf46b5fcbbefab
```

vme64x-core (svec carrier only)

```
repo : git://ohwr.org/hdl-core-lib/vme64x-core.git
commit: b2fc3ce76485404f831d15f7ce31fdde08e234d5
```

3 Architecture

This chapter describes the internal blocks of the FPGA for both SPEC (PCIe) and SVEC (VME64x) carriers. The gateware is designed around one or several Wishbone[1] bus interconnects.

3.1 SPEC (PCIe carrier)

In the PCIe version of the gateware, all blocks (except the memory controller) are connected to the PCIe bridge interface using the same Wishbone bus (*main* bus). The ADC samples are written and read to/from the DDR memory using separate Wishbone bus interconnects. Due to its size, the DDR memory is not mapped on the *main* Wishbone bus and can only be accessed through DMA. Figure 3.1 illustrates the fmc-adc gateware architecture on the SPEC carrier. A crossbar from the general-cores[2] library is used to map the slaves in the Wishbone address space.

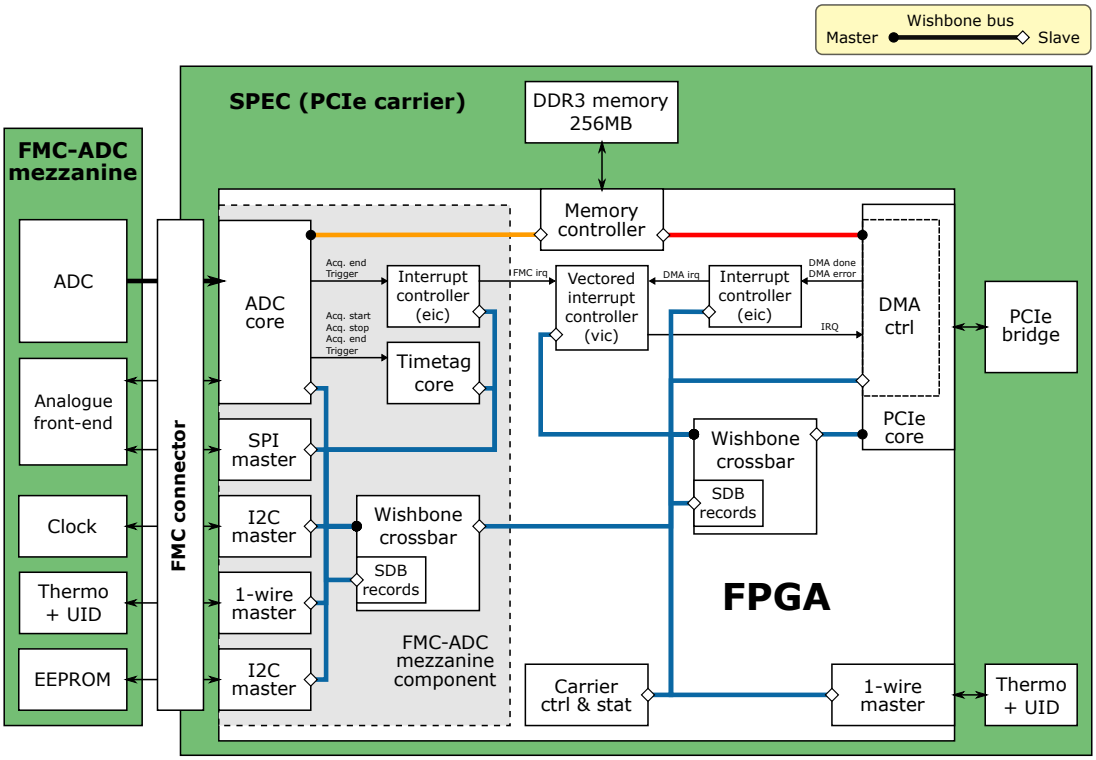


Figure 3.1: FMC-ADC gateware architecture on SPEC carrier.

There are three different Wishbone bus interconnects in the design.

Mapped WB bus (blue)
This bus connects all the peripherals 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).

[1] `http://opencores.org/opencores,wishbone`
[2] `http://www.ohwr.org/projects/general-cores`

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).

Table 3.1 shows the Wishbone slaves mapping and hierarchy. The first column represents the byte address offset from the start of the Wishbone address space (BAR 0).

```
0x0000 crossbar (sdb records)
0x1000 |-- dma controller
0x1100 |-- onewire master
0x1200 |-- spec csr
0x1300 |-- vic
0x1400 |-- dma eic
0x2000 |-- bridge (fmc slot 1) -> crossbar (sdb records)
0x3000 |                          |-- i2c master
0x3100 |                          |-- spi master
0x3200 |                          |-- i2c master
0x3300 |                          |-- adc csr
0x3400 |                          |-- onewire
0x3500 |                          |-- fmc-adc eic
0x3600 |                          |-- timetag core
```

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 gateware, 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 on the SPEC carrier.

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 = bcd encoded release date, format yyyymmdd
name = "spec_fmcadc100m14b"

Top module repository url
repo_url = "fmc-projects/fmc-adc-100m14b4cha/fmc-adc-100m14b4cha-gw.git"
(This is not the full URL, it lacks the "git://ohwr.org/" prefix due to the 63-byte limitation of the field.)

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 = bcd encoded synthesis date, format yyyymmdd
syn_username = username of person who synthesised the design

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.

[3] `http://www.ohwr.org/projects/fpga-config-space`
[4] `http://opencores.org/`

3.1.1 Clock Domains

The SPEC version of the fmc-adc design has five different clock domains. They are listed in the following 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)

3.1.2 GN4124 Core

This block is the interface between the GN4124[5] local bus and the other blocks in the FPGA. The GN4124 is a four lane PCI Express Generation 1.1 bridge. In addition to the 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. BAR4 (Base Address Register) allows access to the GN4124 internal registers. BAR0 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. In the fmc-adc gateware, the master port is connected to the DDR memory controller. The GN4124 Wishbone interfaces (masters and slave) are 32-bit data width and 32-bit word aligned addresses.

Note: It would not be beneficial 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 efficient when reading non-consecutive addresses.

3.1.3 DMA Embedded Interrupt Controller (EIC)

The DMA EIC gathers the interrupts from the GN4124 DMA controller. There are two inputs to the DMA EIC:

- **DMA done**: This interrupt signals the end of a DMA transfer.
- **DMA error**: This interrupt signals an error in a DMA transfer.

The two inputs are multiplexed and the result is forwarded to the VIC (Section 3.3.3 [Vectored Interrupt Controller (VIC)], page 10). Interrupt sources can be masked using the enable and disable registers. An interrupt is cleared by writing a one to the corresponding bit of the status register.

The registers description can be found in annex ([DMA Embedded Interrupt Controller Registers], page 40).

3.1.4 SPEC Carrier Control and Status Registers

This block contains control and status registers related to the SPEC 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 two carrier's LEDs on the front panel.

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.

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

The registers description can be found in annex [SPEC Carrier Registers], page 51.

3.2 SVEC (VME64x carrier)

In the VME64x version of the gateware, all blocks are connected to the VME64x core using a single Wishbone bus. Here the DDR memory is not accessed through DMA, but using a indirect addressing scheme explained later in Section 3.3.2 [DDR Memory Controller], page 10. A crossbar from the general-cores[6] library is used to map the slaves in the Wishbone address space.

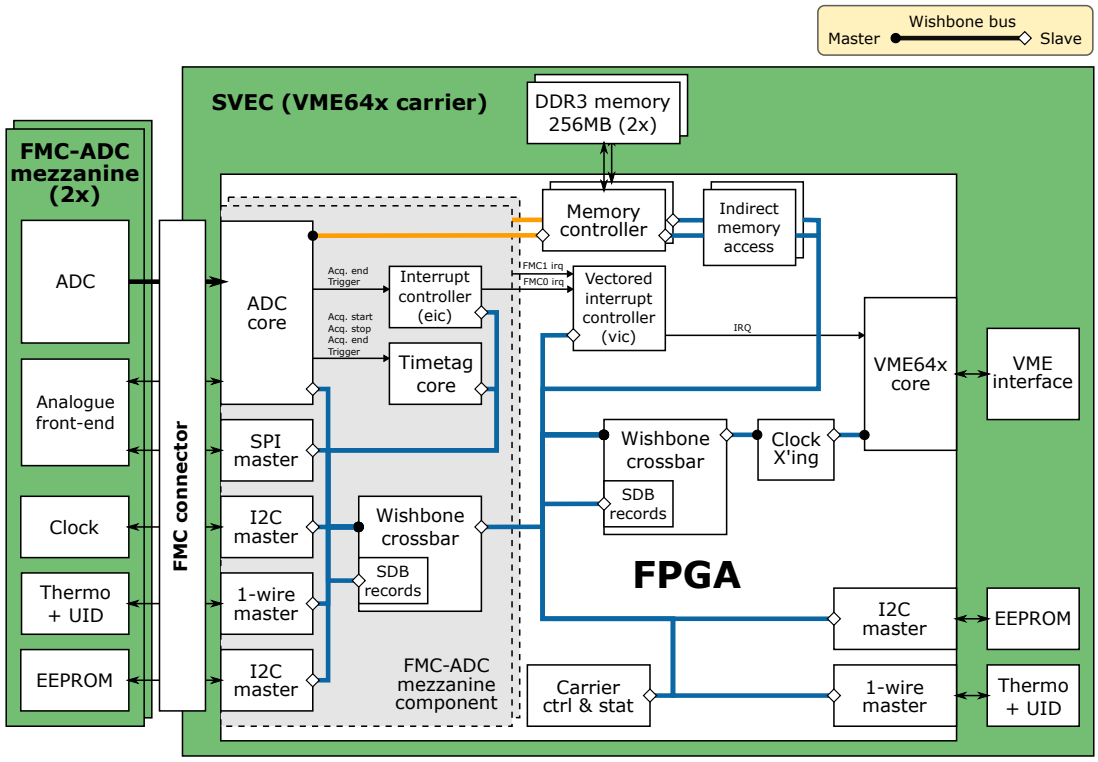


Figure 3.2: FMC-ADC gateware architecture on SVEC carrier.

There are three different Wishbone bus interconnects in the design.

Mapped WB bus (blue)

This bus connects all the peripheral to the VME64x core.
Data: 32-bit, address: 32-bit (word aligned),
Clock: system clock (125MHz) and system clock / 2 (62.5MHz), see note below.

ADC cores to memory controllers (2x, orange)

These two buses are used to write samples from the ADC cores to the DDR memories.
Data: 64-bit, address: 32-bit (word aligned), clock: system clock (125MHz).

Note: The VME64x core cannot work with a clock frequency as high as 125MHz, therefore it is clocked with half the system clock frequency. As the fmc-adc core needs 125MHz to work properly, a Wishbone clock crossing component is inserted between the VME64x core and the

[6] `http://www.ohwr.org/projects/general-cores`

first Wishbone crossbar component. With this topology, only the VME64x core runs at a lower frequency.

Table 3.2 shows the Wishbone slaves mapping and hierarchy. The first column represents the byte address offset from the start of the Wishbone address space.

```
0x0000 crossbar (sdb records)
0x1000 |-- i2c
0x1100 |-- onewire
0x1200 |-- svec csr
0x1300 |-- vic
0x2000 |-- bridge (fmc slot 1) -> crossbar (sdb records)
0x3000 |                          |-- i2c
0x3100 |                          |-- spi
0x3200 |                          |-- i2c
0x3300 |                          |-- adc csr
0x3400 |                          |-- onewire
0x3500 |                          |-- fmc_eic
0x3600 |                          |-- timetag
0x4000 |-- ddr_addr (fmc slot 1)
0x5000 |-- ddr_data (fmc slot 1)
0x6000 |-- bridge (fmc slot 2) -> crossbar (sdb records)
0x7000 |                          |-- i2c
0x7100 |                          |-- spi
0x7200 |                          |-- i2c
0x7300 |                          |-- adc csr
0x7400 |                          |-- onewire
0x7500 |                          |-- fmc_eic
0x7600 |                          |-- timetag
0x8000 |-- ddr_addr (fmc slot 2)
0x9000 |-- ddr_data (fmc slot 2)
```

Table 3.2: Wishbone bus memory mapping.

Same as in the SPEC version, SDB meta-information records are used to identify the gateware. Below is a description of the fields and their content in the fmc-adc design on SVEC carrier.

Integration

vendor_id = 0x0000CE42 (CERN vendor ID)
device_id = 0x47C786A2 (echo "svec_fmc-adc-100m14b4cha"|md5sum|cut -c1-8)
version = [31:16]=major, [15:0]=minor, bcd encoded
date = bcd encoded release date, format yyyymmdd
name = "svec_fmcadc100m14b"

Top module repository url

repo_url = "fmc-projects/fmc-adc-100m14b4cha/fmc-adc-100m14b4cha-gw.git"
(This is not the full URL, it lacks the "git://ohwr.org/" prefix due to the 63-byte limitation of the field.)

Synthesis tool information

syn_module_name = "svec_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 = bcd encoded synthesis date, format yyyymmdd
syn_username = username of person who synthesised the design

3.2.1 Clock Domains

The SPEC version of the fmc-adc design has five different clock domains. They are listed in the following table.

Name	Description	Frequency	Source
`sys_clk_125`	Main system clock	125.00 MHz	20MHz TCXO (carrier)
`sys_clk_62_5`	System clock / 2	62.50 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)

3.2.2 VME64x Core

The VME64x core implements a VME slave on one side and a Wishbone pipelined master on the other side. For more information about the VME64x core, visit the OHWR page[7].

3.2.3 SVEC Carrier Control and Status

This block contains control and status registers related to the SVEC carrier board. The registers description can be found in annex ([SVEC Carrier Registers], page 53).

3.2.4 SVEC Carrier I2C Master

The I2C master accesses the 24AA64 64Kb EEPROM memory chip[8] located on the SVEC board. This memory is useful to store board specific data (e.g. MAC address, White Rabbit calibration data). This block is based on an OpenCores design[9].

I2C slave address	Peripheral
`0x51`	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
```

3.3 Common Cores

3.3.1 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[10].

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
```

[7] `http://www.ohwr.org/projects/vme64x-core`
[8] `http://ww1.microchip.com/downloads/en/devicedoc/21189f.pdf`
[9] `http://opencores.org/project,i2c`
[10] `http://opencores.org/project,sockit_owm`

3.3.2 DDR Memory Controller

The memory controller block is the interface between the 256MB DDR3 memory located on the carrier boards and the other blocks in the FPGA. It is basically a MCB core (Memory Controller Block) generated with Xilinx CoreGen[11] and an additional wrapper implementing two Wishbone slave interfaces[12].

One of the Wishbone slave interfaces is connected to the ADC core. In the SPEC gateware, the other Wishbone slave interface is connected to the DMA Wishbone master of the GN4124 core. In the SVEC gateware, the other slave Wishbone interface is connected to an indirect addressing block.

This block consists of an address pointer register and a data FIFO. To access the DDR memory, the gateware sets the address pointer and then reads/writes data using the FIFO. On each access to the FIFO, the address pointer is automatically incremented.

WB Slave	Description	Data width	Access type
0	ADC core	64-bit	Write only
1	host side	32-bit	Read/write

The memory controller side connected to the chip is 16-bit wide, clocked at 333.33 MHz DDR. This gives a maximum bandwidth of 1333.33 MB/s. Each of the four ADC channels requires 200 MB/s (16-bits per sample, 100 MHz), for a total of 800 MB/s.

In the current design, the two Wishbone ports have the same priority and the arbitration is done with a simple round-robin. Therefore, samples stored in the DDR memory should not be read during an acquisition.

3.3.3 Vectored Interrupt Controller (VIC)

In order to redirect interrupts from different cores to the corresponding driver in the Linux kernel in a generic way, a two layers scheme is used. The first layer is the Embedded Interrupt Controllers (EIC) in each core multiplexing interrupt sources to a single line. The second layer is the Vectored Interrupt Controller (VIC) multiplexing the interrupt lines from the EICs into a single line to the host. The VIC keeps a table, initialized with the base addresses of the EICs connected to each of the input.

Note that the VIC configuration is different between the SPEC and SVEC carriers. The SPEC uses an edge sensitive scheme while the SVEC uses a level sensitive scheme.

SPEC, VIC control register:

enable = 1
polarity = 1
emulate edge sensitive = 1
edge emulation pulse = 750

SVEC, VIC control register:

enable = 1
polarity = 1

Note: On the SPEC carrier, the VIC interrupt request output is connected to GPIO 8 of the GN4124 chip. Therefore, the GN4124 must be configured to generate an MSI when a rising edge is detected on GPIO 8.

The registers description can be found in annex [Vectored Interrupt Controller], page 42).

[11] `http://www.xilinx.com/support/documentation/user_guides/ug388.pdf`

[12] `http://www.ohwr.org/projects/ddr3-sp6-core`

3.4 FMC-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 internal components.

The internal detailed functioning of this block is described further in the document (See Chapter 4 [Configuration], page 14, Chapter 5 [Calibration], page 19, and Chapter 6 [Acquisition], page 21).

3.4.1 Sampling clock

The sampling frequency is determined by a Si570 programmable oscillator located on the fmc-adc mezzanine. By default, the sampling clock is 100MHz (oscillator factor default value), but it can be changed to any frequency from 10MHz to 105MHz. The lower bound is defined by the Si570 oscillator while the upper bound is limited by the LTC2174 ADC itself.

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).

Note: The internal PLL expects a 400MHz input frequency (define in the hdl code), therefore the sampling frequency has to be 100MHz and can't be changed dynamically.

The ADC core implements a sampling clock frequency meter. The measured frequency (in Hz) can be read via a register (see [ADC Core Registers], page 28).

3.4.2 Time-tagging Core

This block allows time-tagging of important events in the ADC core. It is based on two free-running counters; a seconds counter and a 125MHz system clock ticks counter. The system clock ticks counter is also called coarse counter. These two counters are accessible in read/write via a 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.

A time-tag is made of four 32-bit words; meta-data, seconds, coarse, fine. The fine field is always set to zero and the meta-data register does not contain useful information, only random data for debugging purposes.

The following events are time-tagged:

- Trigger
- Acquisition start
- Acquisition stop
- Acquisition end

Note: The trigger time tag corresponds to the moment when the acquisition state machine leaves the `WAIT_TRIG` state.

Note: The trigger time-tag is also stored in the data memory, after the post-trigger samples. This allows to always have a trigger time-tag, even in multi-shot mode (retrieving the time-tag using the trigger interrupt was not fast enough in certain cases).

Note: If during an acquisition no stop command is issued (normal case), the acquisition stop time-tag is not updated.

The register description can be found in annex ([Time-tagging Core Registers], page 45).

3.4.3 FMC-ADC Control and Status Registers

This block contains control and status registers related to the fmc-adc core. The registers description can be found in annex ([ADC Core Registers], page 28).

3.4.4 Mezzanine SPI Master

This SPI master controls the LTC2174 ADC[13] and the four MAX5442 offset DACs[14]. The following table shows how the peripherals are wired to the core. This block is based on an OpenCores design[15].

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)
```

3.4.5 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[16].

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

3.4.6 Mezzanine I2C Master

This I2C master controls the Si570 programmable oscillator chip[17] located on the mezzanine board. This chip is used to produce the ADC sampling clock. This block is based on an OpenCores design[18].

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
```

3.4.7 Mezzanine System Management I2C Master

This I2C master accesses the 24AA64 64Kb EEPROM memory chip[19] located on the mezzanine board. This memory is mandatory as specified in the FMC standard (VITA 57.1). It is connected

[13] http://cds.linear.com/docs/en/datasheet/21754314fa.pdf
[14] http://datasheets.maximintegrated.com/en/ds/MAX5441-MAX5444.pdf
[15] http://opencores.org/project,spi
[16] http://opencores.org/project,sockit_owm
[17] https://www.silabs.com/Support%20Documents/TechnicalDocs/si570.pdf
[18] http://opencores.org/project,i2c
[19] http://ww1.microchip.com/downloads/en/devicedoc/21189f.pdf

to the system management I2C bus, also specified in the FMC standard. This block is based on an OpenCores design[20].

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
```

3.4.8 FMC-ADC Embedded Interrupt Controller (EIC)

The fmc-adc EIC gathers the interrupts from the ADC core. There are two inputs to the fmc-adc EIC:

- **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.

The two inputs are multiplexed and the result is forwarded to the VIC (Section 3.3.3 [Vectored Interrupt Controller (VIC)], page 10). Interrupt sources can be masked using the enable and disable registers. An interrupt is cleared by writing a one to the corresponding bit of the status register.

The registers description can be found in annex ([FMC-ADC Embedded Interrupt Controller Registers], page 38).

[20] `http://opencores.org/project,i2c`

4 Configuration

Figure 4.1 is a block diagram of the ADC core part in the sampling clock domain. It contains an 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).

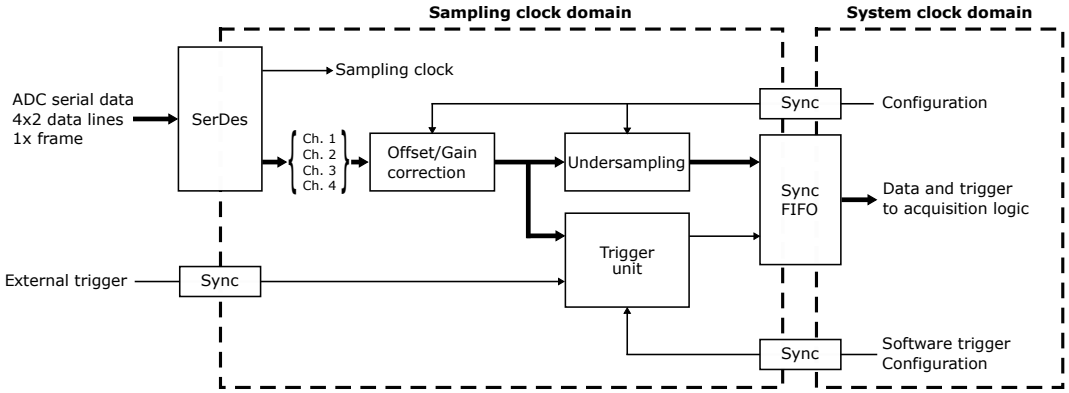


Figure 4.1: ADC core diagram (sampling clock domain).

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

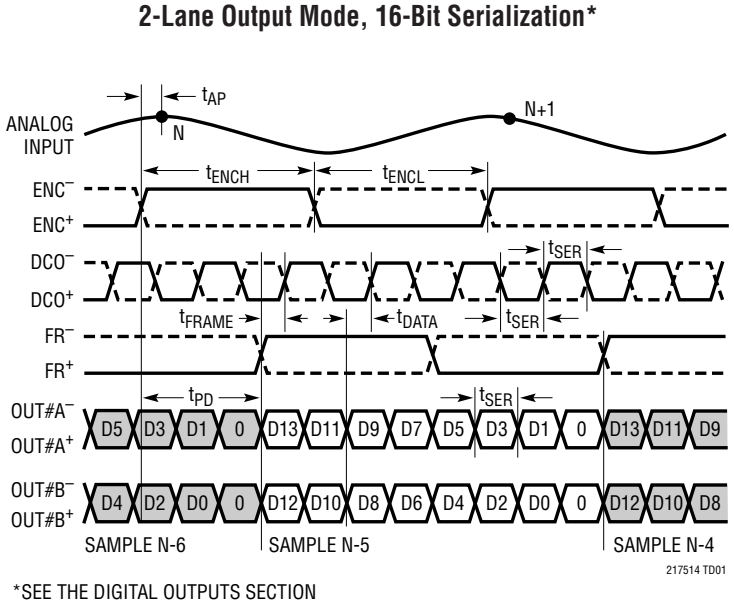


Figure 4.2: LTC2174 data output mode waveforms.

There are two 800Mbit/s lanes per ADC channel. Eight data lanes in total 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 shown in Figure 4.2, the two LSB bits of a data word are always set to zero.

4.1 Control and Status Registers

Writing one to to the `FMC_CLK_OE` field of the ADC core control register enables 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

Figure 4.3 shows a simplified schematic diagram of the analogue input stage used for each channel. Each input can be independently configured with one of the three available ranges; 100mV, 1V, 10V. Each range is defined as the maximum peak-to-peak input voltage. Independently 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 are used to apply the various configurations. They are represented by standard switch symbols in the simplified schematic.

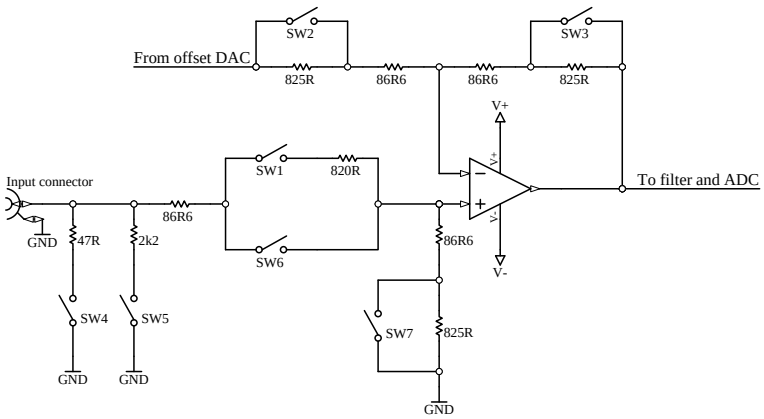


Figure 4.3: Simplified schematic diagram of the analogue input stage.

Only the following input switch configurations are valid. For all others switch configurations, the behavior 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 two hardware and one software sources. The hardware and software paths can be enabled independently. The two paths are then or'ed together to drive a delay generator. The delay generator allows to insert a predefined number of sampling clock periods before the trigger is forwarded to the acquisition state machine. Figure 4.4 shows a simplified digram of the trigger unit.

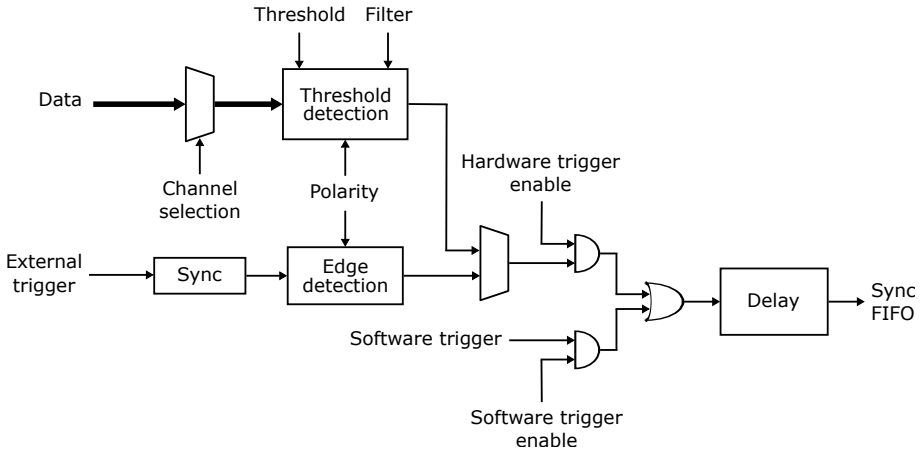


Figure 4.4: Trigger unit diagram.

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 polarity 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.

To use the internal trigger source, both the ADC input channel and the threshold should be configured. By default, channel 1 is selected and the threshold is set to 0. Note that the threshold is 16-bit signed (two's complement). Figure 4.5 sketches the internal hardware trigger threshold behavior.

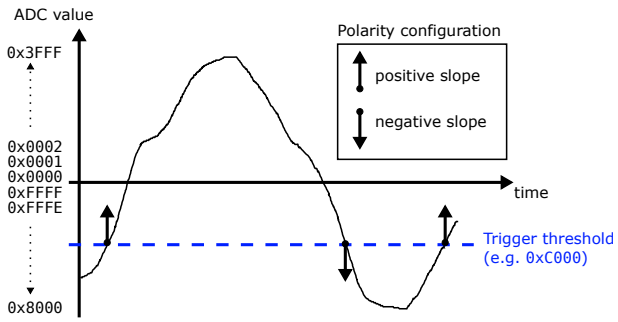


Figure 4.5: Internal hardware trigger threshold.

Furthermore, a glitch filter can be applied to the threshold detection. The glitch filter is useful to trigger on noisy signals. In order to help setting the glitch filter, an internal trigger test mode can be activated. When the test mode is enabled, data from channels 2, 3 and 4 is replaced as follow:

Channel 2 Input signal over threshold
Channel 3 Input signal over threshold filtered
Channel 4 Trigger

The software trigger source consists of a pulse generated when a write cycle is detected on the *Software trigger* register. For further information on the trigger configuration registers see [ADC Core Registers], page 28.

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 gateware.

5 Calibration

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 an 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-FRU	binary	IPMI records
`0x100`	calib	binary	Calibration values
`auto`	name	ascii	Contains "adc_100m"
`0x800`	data	binary	Empty directory
`0x200`	.	binary	Root 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.

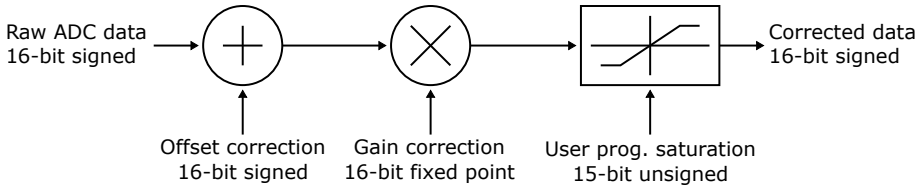


Figure 5.1: ADC offset and gain correction block.

[1] `http://www.ohwr.org/attachments/download/1594/sdbfs-2012-09-19.pdf`

[2] Platform Management FRU Information Storage Definition v1.0

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:

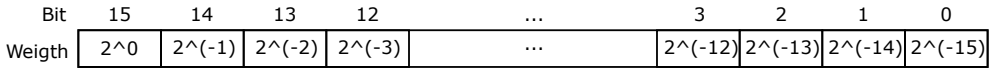


Figure 5.2: ADC gain register format.

After the offset and gain corrections are applied, the signal is saturated to a user-programmable value. One register per channel allows to set the saturation value. The saturation register takes a 15-bit unsigned value. From this value, two 'symmetrical' 16-bit signed numbers are derived and taken as the saturation boundaries.

Note: Because the default value (on FPGA start-up) is not configurable in wbgen2, the gain, offset and saturation registers are set to 0x0 at start-up. Therefore, the driver has to initialise those registers.

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 writing 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.

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 23, and Section 6.2 [Multi-shot Mode], page 24. 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).

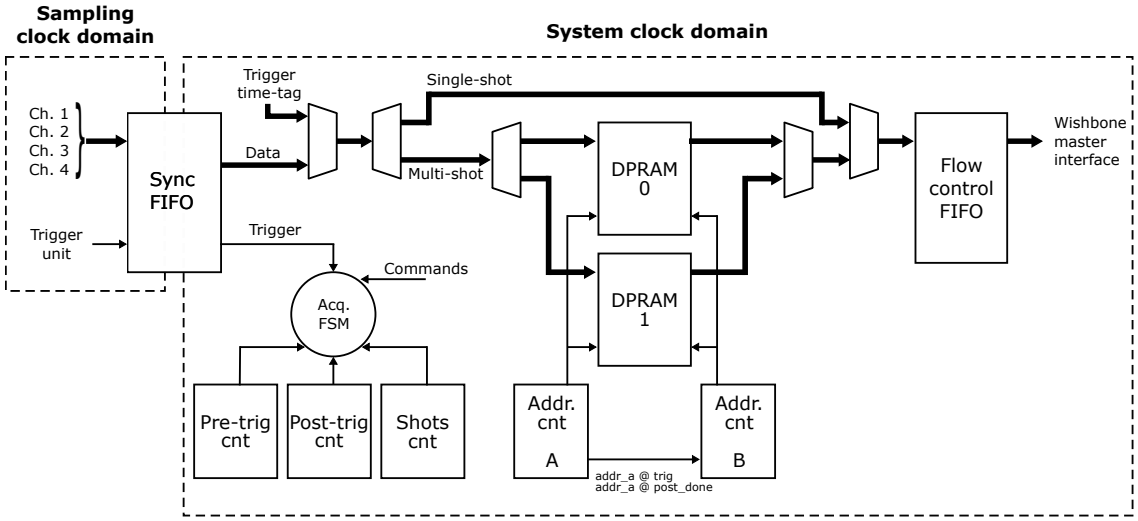


Figure 6.1: Acquisition logic diagram (system clock domain).

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

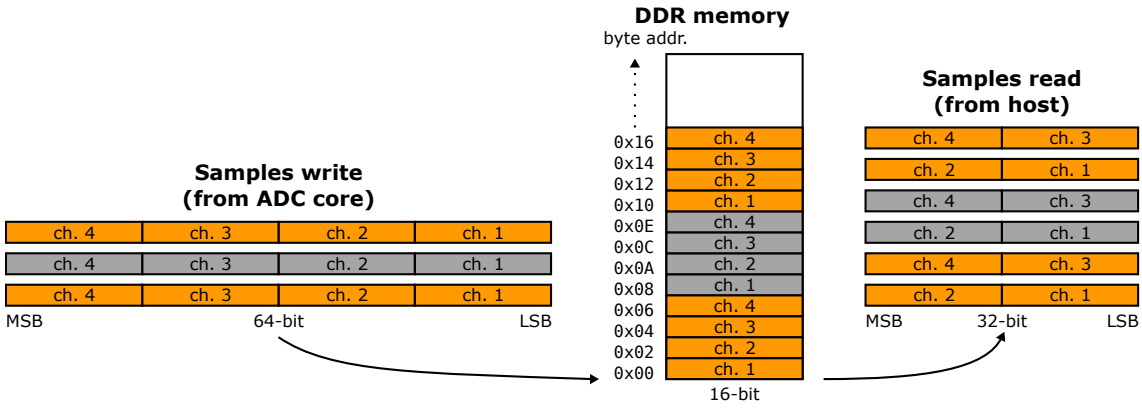


Figure 6.2: Illustration of samples storage in DDR memory.

The acquisition process is driven by a state machine. Figure 6.3 represents its states and transitions. At start-up (system reset), the state machine is IDLE, waiting for an acquisition 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 28).

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. If the number of programmed pre-trigger samples is zero, the state machine skips the PRE_TRIG state and it foes directly to WAIT_TRIG. When a valid trigger is detected, the state machine moves to POST_TRIG. It will stay in this state until the programmed number of post-trigger samples is reached. The next state is TRIG_TAG where the trigger time-tag (4x 32-bit word) is pushed after the last post-trigger sample (to be stored in DDR memory). When the trigger time-tag has been pushed (two clock cycles), the state machine goes to DECR_SHOT. From DECR_SHOT it either goes back to IDLE if the number of shots is reached or it repeats the same cycle for the next shot.

When the acquisition is finished (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: Triggers 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.

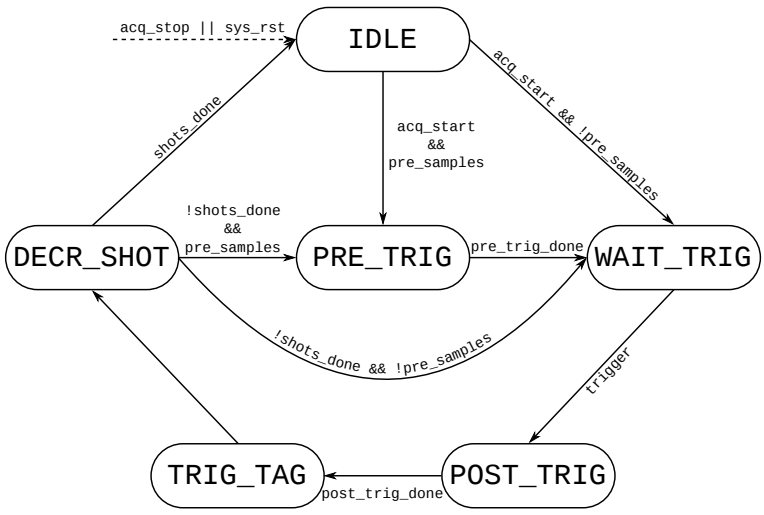


Figure 6.3: Acquisition state machine.

There are two LEDs 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: The number of pre-trigger sample can be zero, but there **must** be at least one post-trigger sample.

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

Note: The start of an acquisition is prohibited if either the number of shots or the number of post-trigger samples is equal to zero.

6.1 Single-shot Mode

The procedure below lists the different steps 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 (if enabled).
4. At the end of the acquisition, another interrupt is generated.
5. Read trigger position register.
6. Configure the DMA to retrieve 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 directly 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 overwrites previous samples. In order to allow the software to retrieve the requested samples (around the trigger), the *Trigger address* register stores the write pointer address at the trigger moment.

Note: The value stored in the *Trigger address* register is a byte address.

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

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

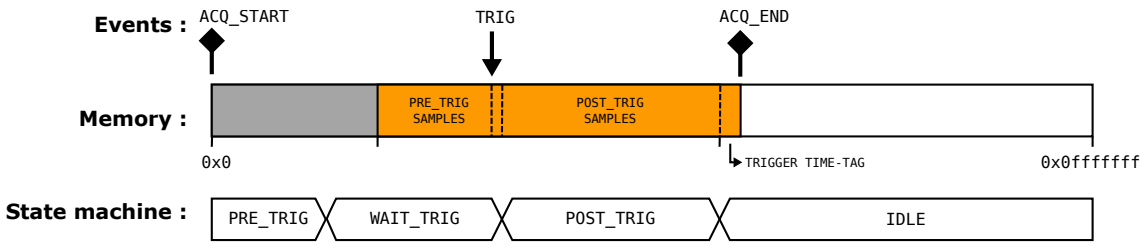


Figure 6.4: Single-shot mode acquisition example.

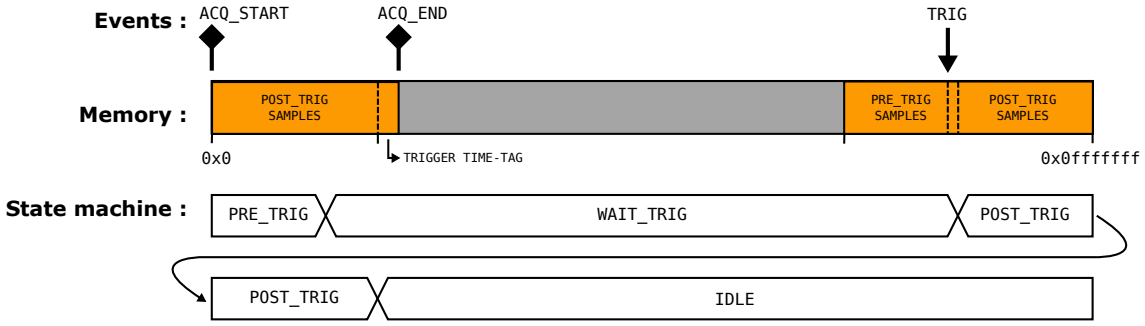


Figure 6.5: Single-shot mode acquisition example (overlapping DDR memory).

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 times as the number of configured shots. This means that if the board is configured for N shots, it will generate N trigger interrupts (if enabled) and then another interrupt at the end of the acquisition. A counter, accessible via a register, shows the remaining number of shots (see [ADC Core Registers], page 28).

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 use dpram0 and odd shots dpram1.

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

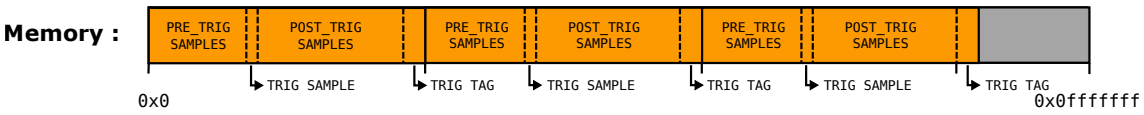


Figure 6.6: DDR memory usage in multi-shot mode acquisition.

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) + 2 (time-tag).

Note: In multi-shot mode, the start of an acquisition is prohibited if the number of sample per shot is bigger or equal to the dpram size.

Note: The size of the dprams is configurable during the generation of the FPGA bitstream (VHDL generic), but not at runtime. The software can retrieve the maximum *allowed* value from the *Multi-shot sample depth register* (see [ADC Core Registers], page 28). The value stored in that read-only register already takes into account the 2 samples reserved for the time-tag (eg. if the actual maximum number of samples allowed is 8000, the register will read 7998).

7 Missing Features and Improvements

An up-to-date list of known bugs, missing features and improvements is available in the OHWR project page:
`http://www.ohwr.org/projects/fmc-adc-100m14b4cha-gw/issues`

A roadmap for future releases is also available in the OHWR project page:
`http://www.ohwr.org/projects/fmc-adc-100m14b4cha-gw/roadmap`

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 (calib 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 (calib file).

Appendix B ADC Core Registers

The registers documentation have been generated using `wbgen2`[1].

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	shots_cnt	Remaining shots counter
0x1c	REG	trig_pos	Trigger address register
0x20	REG	fs_freq	Sampling clock frequency
0x24	REG	sr	Sample rate
0x28	REG	pre_samples	Pre-trigger samples
0x2c	REG	post_samples	Post-trigger samples
0x30	REG	samples_cnt	Samples counter
0x34	REG	ch1_ctl	Channel 1 control register
0x38	REG	ch1_sta	Channel 1 status register
0x3c	REG	ch1_gain	Channel 1 gain calibration register
0x40	REG	ch1_offset	Channel 1 offset calibration register
0x44	REG	ch1_sat	Channel 1 saturation register
0x48	REG	ch2_ctl	Channel 2 control register
0x4c	REG	ch2_sta	Channel 2 status register
0x50	REG	ch2_gain	Channel 2 gain calibration register
0x54	REG	ch2_offset	Channel 2 offset calibration register
0x58	REG	ch2_sat	Channel 2 saturation register
0x5c	REG	ch3_ctl	Channel 3 control register
0x60	REG	ch3_sta	Channel 3 status register
0x64	REG	ch3_gain	Channel 3 gain calibration register
0x68	REG	ch3_offset	Channel 3 offset calibration register
0x6c	REG	ch3_sat	Channel 3 saturation register
0x70	REG	ch4_ctl	Channel 4 control register
0x74	REG	ch4_sta	Channel 4 status register
0x78	REG	ch4_gain	Channel 4 gain calibration register
0x7c	REG	ch4_offset	Channel 4 offset calibration register
0x80	REG	ch4_sat	Channel 4 saturation register
0x84	REG	multi_depth	Multi-shot sample depth register

[1] `http://www.ohwr.org/projects/wishbone-gen`

B.2 ctl - Control register

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

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

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
5	R/O	ACQ_CFG	X	Acquisition configuration status

Field	Description
fsm	States: 0: illegal 1: IDLE 2: PRE_TRIG 3: WAIT_TRIG 4: POST_TRIG 5: TRIG_TAG 6: DECR_SHOT 7: illegal
serdes_pll	Sampling clock recovery PLL. 0: not locked 1: locked
serdes_synced	0: bitslip in progress 1: serdes synchronized

acq_cfg	0: Unauthorised acquisition configuration (will prevent acquisition to start) 1: Valid acquisition configuration • Shot number > 0 • Post-trigger sample > 0

B.4 trig_cfg - Trigger configuration

Bits	Access	Prefix	Default	Name
0	R/W	HW_TRIG_SEL	0	Hardware trigger selection
1	R/W	HW_TRIG_POL	0	Hardware trigger polarity
2	R/W	HW_TRIG_EN	0	Hardware trigger enable
3	R/W	SW_TRIG_EN	0	Software trigger enable
5...4	R/W	INT_TRIG_SEL	0	Channel selection for internal trigger
6	R/W	INT_TRIG_TEST_EN	0	Enable internal trigger test mode
7	R/W	RESERVED	0	Reserved
15...8	R/W	INT_TRIG_THRES_FILT	0	Internal trigger threshold glitch filter
31...16	R/W	INT_TRIG_THRES	0	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
int_trig_test_en	Test mode: ch1 = Channel 1 input(analogue) ch2 = Channel input over threshold (digital) ch3 = Channel input over threshold filtered (digital) ch4 = Trigger (digital)
reserved	Ignore on read, write with 0's
int_trig_thres_filt	Configures the internal trigger threshold glitch filter length.
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	0	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	0	Software trigger (ignore on read)

B.7 shots - Number of shots

Bits	Access	Prefix	Default	Name
15...0	R/W	NB	0	Number of shots

Field	Description
nb	Number of shots required in multi-shot mode, set to one for single-shot mode.

B.8 shots_cnt - Remaining shots counter

Bits	Access	Prefix	Default	Name
15...0	R/O	VAL	X	Remaining shots counter

Field	Description
val	Counts the number of remaining shots to acquire.

B.9 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.10 fs_freq - Sampling clock frequency

Bits	Access	Prefix	Default	Name
31...0	R/O	FS_FREQ	X	Sampling clock frequency

Field	Description
fs_freq	ADC sampling clock frequency in Hz

B.11 sr - Sample rate

Bits	Access	Prefix	Default	Name
31...0	R/W	UNDERSAMPLE	0	Undersampling ratio

Field	Description
undersample	Undersampling ratio. Takes one sample every N samples and discards the others (N = undersampling ratio).

B.12 pre_samples - Pre-trigger samples

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

Field	Description
pre_samples	Number of requested pre-trigger samples (>1).

B.13 post_samples - Post-trigger samples

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

Field	Description
post_ samples	Number of requested post-trigger samples (>1).

B.14 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.15 ch1_ctl - Channel 1 control register

Bits	Access	Prefix	Default	Name
6...0	R/W	SSR	0	Solid state relays control for channel 1

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.

B.16 ch1_sta - Channel 1 status register

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

Field	Description
val	Current ADC raw value. The format is binary two's complement.

B.17 ch1_gain - Channel 1 gain calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	0	Gain calibration for channel 1

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)

B.18 ch1_offset - Channel 1 offset calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	0	Offset calibration for channel 1

Field	Description
val	Offset applied to all data coming from the ADC. The format is binary two's complement.

B.19 ch1_sat - Channel 1 saturation register

Bits	Access	Prefix	Default	Name
14...0	R/W	VAL	0	Saturation value for channel 1

Field	Description
val	Saturation applied to all data coming from the offset/gain correction block. The format is 15-bit unsigned.

B.20 ch2_ctl - Channel 2 control register

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

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.

B.21 ch2_sta - Channel 2 status register

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

Field	Description
val	Current ADC raw value. The format is binary two's complement.

B.22 ch2_gain - Channel 2 gain calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	0	Gain calibration for channel 2

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)

B.23 ch2_offset - Channel 2 offset calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	0	Offset calibration for channel 2

Field	Description
val	Offset applied to all data coming from the ADC. The format is binary two's complement.

B.24 ch2_sat - Channel 2 saturation register

Bits	Access	Prefix	Default	Name
14...0	R/W	VAL	0	Saturation value for channel 2

Field	Description
val	Saturation applied to all data coming from the offset/gain correction block. The format is 15-bit unsigned.

B.25 ch3_ctl - Channel 3 control register

Bits	Access	Prefix	Default	Name
6...0	R/W	SSR	0	Solid state relays control for channel 3

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.

B.26 ch3_sta - Channel 3 status register

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

Field	Description
val	Current ADC raw value. The format is binary two's complement.

B.27 ch3_gain - Channel 3 gain calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	0	Gain calibration for channel 3

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)

B.28 ch3_offset - Channel 3 offset calibration register

Bits	Access	Prefix	Default	Name
15...0	R/W	VAL	0	Offset calibration for channel 3

Field	Description
val	Offset applied to all data coming from the ADC. The format is binary two's complement.

B.29 ch3_sat - Channel 3 saturation register

Bits	Access	Prefix	Default	Name
14...0	R/W	VAL	0	Saturation value for channel 3

Field	Description
val	Saturation applied to all data coming from the offset/gain correction block. The format is 15-bit unsigned.

B.30 ch4_ctl - Channel 4 control register

Bits	Access	Prefix	Default	Name
6...0	R/W	SSR	0	Solid state relays control for channel 4

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.

B.31 ch4_sta - Channel 4 status register

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

Field	Description
val	Current ADC raw value. The format is binary two's complement.

B.32 ch4_gain - Channel 4 gain calibration register

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

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)

B.33 ch4_offset - Channel 4 offset calibration register

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

Field	Description
val	Offset applied to all data coming from the ADC. The format is binary two's complement.

B.34 ch4_sat - Channel 4 saturation register

Bits	Access	Prefix	Default	Name
14...0	R/W	VAL	0	Saturation value for channel 4

Field	Description
val	Saturation applied to all data coming from the offset/gain correction block. The format is 15-bit unsigned.

B.35 multi_depth - Multi-shot sample depth register

Bits	Access	Prefix	Default	Name
31...0	R/O	MULTI_DEPTH	X	Multi-shot sample depth

Field	Description
multi_depth	Maximum sample depth allowed in multi-shot acquisition mode, excluding two samples already reserved for time tag

Appendix C FMC-ADC Embedded Interrupt Controller Registers

The registers documentation have been generated using `wbgen2`[1].

C.1 Memory map summary

Address	Type	Prefix	Name
0x0	REG	EIC_IDR	Interrupt disable register
0x4	REG	EIC_IER	Interrupt enable register
0x8	REG	EIC_IMR	Interrupt mask register
0xc	REG	EIC_ISR	Interrupt status register

C.2 EIC_IDR - Interrupt disable register

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

Bits	Access	Prefix	Default	Name
0	W/O	TRIG	0	Trigger interrupt
1	W/O	ACQ_END	0	End of acquisition interrupt

Field	Description
trig	write 1: disable interrupt 'Trigger interrupt' write 0: no effect
acq_end	write 1: disable interrupt 'End of acquisition interrupt' write 0: no effect

C.3 EIC_IER - Interrupt enable register

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

Bits	Access	Prefix	Default	Name
0	W/O	TRIG	0	Trigger interrupt
1	W/O	ACQ_END	0	End of acquisition interrupt

Field	Description
trig	write 1: enable interrupt 'Trigger interrupt' write 0: no effect
acq_end	write 1: enable interrupt 'End of acquisition interrupt' write 0: no effect

C.4 EIC_IMR - Interrupt mask register

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

Bits	Access	Prefix	Default	Name
0	R/O	TRIG	X	Trigger interrupt

[1] http://www.ohwr.org/projects/wishbone-gen

1	R/O	ACQ_END	X	End of acquisition interrupt

Field	Description
trig	read 1: interrupt 'Trigger interrupt' is enabled read 0: interrupt 'Trigger interrupt' is disabled
acq_end	read 1: interrupt 'End of acquisition interrupt' is enabled read 0: interrupt 'End of acquisition interrupt' is disabled

C.5 EIC_ISR - Interrupt status register

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.

Bits	Access	Prefix	Default	Name
0	R/W	TRIG	X	Trigger interrupt
1	R/W	ACQ_END	X	End of acquisition interrupt

Field	Description
trig	read 1: interrupt 'Trigger interrupt' is pending read 0: interrupt not pending write 1: clear interrupt 'Trigger interrupt' write 0: no effect
acq_end	read 1: interrupt 'End of acquisition interrupt' is pending read 0: interrupt not pending write 1: clear interrupt 'End of acquisition interrupt' write 0: no effect

Appendix D DMA Embedded Interrupt Controller Registers

The registers documentation have been generated using `wbgen2`[1].

D.1 Memory map summary

Address	Type	Prefix	Name
0x0	REG	EIC_IDR	Interrupt disable register
0x4	REG	EIC_IER	Interrupt enable register
0x8	REG	EIC_IMR	Interrupt mask register
0xc	REG	EIC_ISR	Interrupt status register

D.2 EIC_IDR - Interrupt disable register

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

Bits	Access	Prefix	Default	Name
0	W/O	DMA_DONE	0	DMA done interrupt
1	W/O	DMA_ERROR	0	DMA error interrupt

Field	Description
dma_done	write 1: disable interrupt 'DMA done interrupt' write 0: no effect
dma_error	write 1: disable interrupt 'DMA error interrupt' write 0: no effect

D.3 EIC_IER - Interrupt enable register

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

Bits	Access	Prefix	Default	Name
0	W/O	DMA_DONE	0	DMA done interrupt
1	W/O	DMA_ERROR	0	DMA error interrupt

Field	Description
dma_done	write 1: enable interrupt 'DMA done interrupt' write 0: no effect
dma_error	write 1: enable interrupt 'DMA error interrupt' write 0: no effect

D.4 EIC_IMR - Interrupt mask register

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

Bits	Access	Prefix	Default	Name
0	R/O	DMA_DONE	X	DMA done interrupt

[1] http://www.ohwr.org/projects/wishbone-gen

1	R/O	DMA_ERROR	X	DMA error interrupt

Field	Description
dma_done	read 1: interrupt 'DMA done interrupt' is enabled read 0: interrupt 'DMA done interrupt' is disabled
dma_error	read 1: interrupt 'DMA error interrupt' is enabled read 0: interrupt 'DMA error interrupt' is disabled

D.5 EIC_ISR - Interrupt status register

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.

Bits	Access	Prefix	Default	Name
0	R/W	DMA_DONE	X	DMA done interrupt
1	R/W	DMA_ERROR	X	DMA error interrupt

Field	Description
dma_done	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 1: interrupt 'DMA error interrupt' is pending read 0: interrupt not pending write 1: clear interrupt 'DMA error interrupt' write 0: no effect

Appendix E Vectored Interrupt Controller

The registers documentation have been generated using wbgen2[1].

E.1 Memory map summary

Address	Type	Prefix	Name
0x0	REG	CTL	VIC Control Register
0x4	REG	RISR	Raw Interrupt Status Register
0x8	REG	IER	Interrupt Enable Register
0xc	REG	IDR	Interrupt Disable Register
0x10	REG	IMR	Interrupt Mask Register
0x14	REG	VAR	Vector Address Register
0x18	REG	SWIR	Software Interrupt Register
0x1c	REG	EOIR	End Of Interrupt Acknowledge Register
0x20 - 0x3f	MEM	IVT_RAM	Interrupt Vector Table

E.2 CTL - VIC Control Register

Bits	Access	Prefix	Default	Name
0	R/W	ENABLE	0	VIC Enable
1	R/W	POL	0	VIC output polarity
2	R/W	EMU_EDGE	0	Emulate Edge sensitive output
18...3	R/W	EMU_LEN	0	Emulated Edge pulse timer

Field	Description
ENABLE	• 1: enables VIC operation • 0: disables VIC operation
POL	• 1: IRQ output is active high • 0: IRQ output is active low
EMU_EDGE	• 1: Forces a low pulse of EMU_LEN clock cycles at each write to EOIR. Useful for edge-only IRQ controllers such as Gennum. • 0: Normal IRQ master line behavior
EMU_LEN	Length of the delay (in clk_sys_i cycles) between write to EOIR and re-assertion of irq_master_o.

E.3 RISR - Raw Interrupt Status Register

Bits	Access	Prefix	Default	Name
31...0	R/O	RISR	X	Raw interrupt status

Field	Description
RISR	Each bit reflects the current state of corresponding IRQ input line. • read 1: interrupt line is currently active • read 0: interrupt line is inactive

[1] http://www.ohwr.org/projects/wishbone-gen

E.4 IER - Interrupt Enable Register

Bits	Access	Prefix	Default	Name
31...0	W/O	IER	0	Enable IRQ

Field	Description
IER	• write 1: enables interrupt associated with written bit • write 0: no effect

E.5 IDR - Interrupt Disable Register

Bits	Access	Prefix	Default	Name
31...0	W/O	IDR	0	Disable IRQ

Field	Description
IDR	• write 1: enables interrupt associated with written bit • write 0: no effect

E.6 IMR - Interrupt Mask Register

Bits	Access	Prefix	Default	Name
31...0	R/O	IMR	X	IRQ disabled/enabled

Field	Description
IMR	• read 1: interrupt associated with read bit is enabled • read 0: interrupt is disabled

E.7 VAR - Vector Address Register

Bits	Access	Prefix	Default	Name
31...0	R/O	VAR	X	Vector Address

Field	Description
VAR	Address of pending interrupt vector, read from Interrupt Vector Table

E.8 SWIR - Software Interrupt Register

Writing 1 to one of bits of this register causes a software emulation of the respective interrupt.

Bits	Access	Prefix	Default	Name
31...0	W/O	SWIR	0	SWI interrupt mask

E.9 EOIR - End Of Interrupt Acknowledge Register

Bits	Access	Prefix	Default	Name
31...0	W/O	EOIR	0	End of Interrupt

Field	Description
EOIR	Any write operation acknowledges the pending interrupt. Then, VIC advances to another pending interrupt(s) or releases the master interrupt output.

Appendix F Time-tagging Core Registers

The registers documentation have been generated using `wbgen2`[1].

F.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

[1] `http://www.ohwr.org/projects/wishbone-gen`

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)

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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

F.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 G SPEC Carrier Registers

The registers documentation have been generated using `wbgen2`[1].

G.1 Memory map summary

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

G.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

G.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

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.

[1] http://www.ohwr.org/projects/wishbone-gen

G.4 `ctrl` - Control

Bits	Access	Prefix	Default	Name
0	R/W	LED_GREEN	0	Green LED
1	R/W	LED_RED	0	Red LED
2	R/W	DAC_CLR_N	0	DAC clear

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

G.5 `rst` - Reset Register

Controls software reset of the mezzanine including the ddr interface and the time-tagging core.

Bits	Access	Prefix	Default	Name
0	R/W	FMC0_N	X	State of the reset line

Field	Description
fmc0_n	write 0: FMC is held in reset write 1: Normal FMC operation (default)

Appendix H SVEC Carrier Registers

The registers documentation have been generated using `wbgen2`[1].

H.1 Memory map summary

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

H.2 carrier - Carrier type and PCB version

Bits	Access	Prefix	Default	Name
4...0	R/O	PCB_REV	X	PCB revision
15...5	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

H.3 stat - Status

Bits	Access	Prefix	Default	Name
0	R/O	FMC0_PRES	X	FMC 1 presence
1	R/O	FMC1_PRES	X	FMC 2 presence
2	R/O	SYS_PLL_LCK	X	System clock PLL status
3	R/O	DDR0_CAL_DONE	X	DDR3 bank 4 calibration status
4	R/O	DDR1_CAL_DONE	X	DDR3 bank 5 calibration status

Field	Description
fmc0_pres	0: FMC slot 1 is populated 1: FMC slot 1 is not populated.
fmc1_pres	0: FMC slot 2 is populated 1: FMC slot 2 is not populated.
sys_pll_lck	0: not locked 1: locked.

[1] http://www.ohwr.org/projects/wishbone-gen

ddr0_cal_done	0: not done 1: done.
ddr1_cal_done	0: not done 1: done.

H.4 ctrl - Control

Bits	Access	Prefix	Default	Name
15...0	R/W	FP_LEDS_MAN	0	Front panel LED manual control

Field	Description
fp_leds_man	Height front panel LED, two bits per LED. 00 = OFF 01 = Green 10 = Red 11 = Orange

H.5 rst - Reset Register

Controls software reset of the mezzanines including the ddr interface and the time-tagging core.

Bits	Access	Prefix	Default	Name
0	R/W	FMC0_N	X	State of the FMC 1 reset line
1	R/W	FMC1_N	X	State of the FMC 2 reset line

Field	Description
fmc0_n	write 0: FMC is held in reset write 1: Normal FMC operation (default)
fmc1_n	write 0: FMC is held in reset write 1: Normal FMC operation (default)

Appendix I Glossary

I.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.

SDB Self-Describing Bus

VIC Vectored Interrupt Controller

EIC Embedded Interrupt Controller