Cryring BPM
Gateware Documentation

Version 2.1

February 9, 2023

René Geißler
r.geissler@gsi.de

GSI Helmholtzzentrum für Schwerionenforschung GmbH

Contents

Documentation formats

There are three available documentation formats:

- Markdown: https://git.gsi.de/BEA_HDL/Cryring_BPM_Gateware/-/blob/master/README.md
- DokuWiki: https://www-bd.gsi.de/dokuwiki/doku.php?id=ds:projects:cryring:bpm
- LaTeX (PDF): https://www-bd.gsi.de/dokuwiki/lib/exe/fetch.php?media=ds:projects:cryring:bpm:gateware:cryring_bpm_gateware_documentation.pdf

All documentation is based on the content of `README.md`, which is the most up to date format. The DokuWiki and the PDF version have been generated automatically, but will not be updated automatically.

Up to date DokuWiki and PDF versions can be downloaded from the CI/CD section: https://git.gsi.de/BEA_HDL/Cryring_BPM_Gateware/-/pipelines.

Resources

The code of this project and also the source of this documentation are under version control in a Git repository whose upstream is:
https://git.gsi.de/BEA_HDL/Cryring_BPM_Gateware
The relevant branch is *master*.

Additional datasheets and papers are included as a Git submodule of the main Git repository. The upstream of the submodule is:
https://git.gsi.de/BEA_HDL/datasheets
The relevant branch is *master*.

Common code is included as a Git submodule of the main Git repository. The upstream of the submodule is:
https://git.gsi.de/BEA_HDL/FPGA_Common
The relevant branch is *master*.

Installation scripts to set up a Gitlab runner for continuous integration including all necessary software to build the gateware can be found in a Git repository whose upstream is:
https://git.gsi.de/BEA_HDL/Gitlab_Runner_Setup_Centos_7
The relevant branch is *master*.

1 Introduction

This document describes the gateware (= FPGA firmware) implementation of the Beam Position Monitor (BPM) for the Cryring accelerator at GSI. The term Trajectory Measurement System (TMS) is also common for this system and is used as a synonym for BPM. The BPM measures the horizontal and vertical beam positions at nine places of the accelerator ring, resulting in 18 location results.

There had been a previous implementation by Piotr Miedzik, but since no documentation could be found besides a conference paper [2], it was decided to reimplement the gateware.

1.1 Measurement principle

At each of the 18 measurement spots two capacitor plates are used to detect the electrostatic induction of the passing by charged particle bunches.

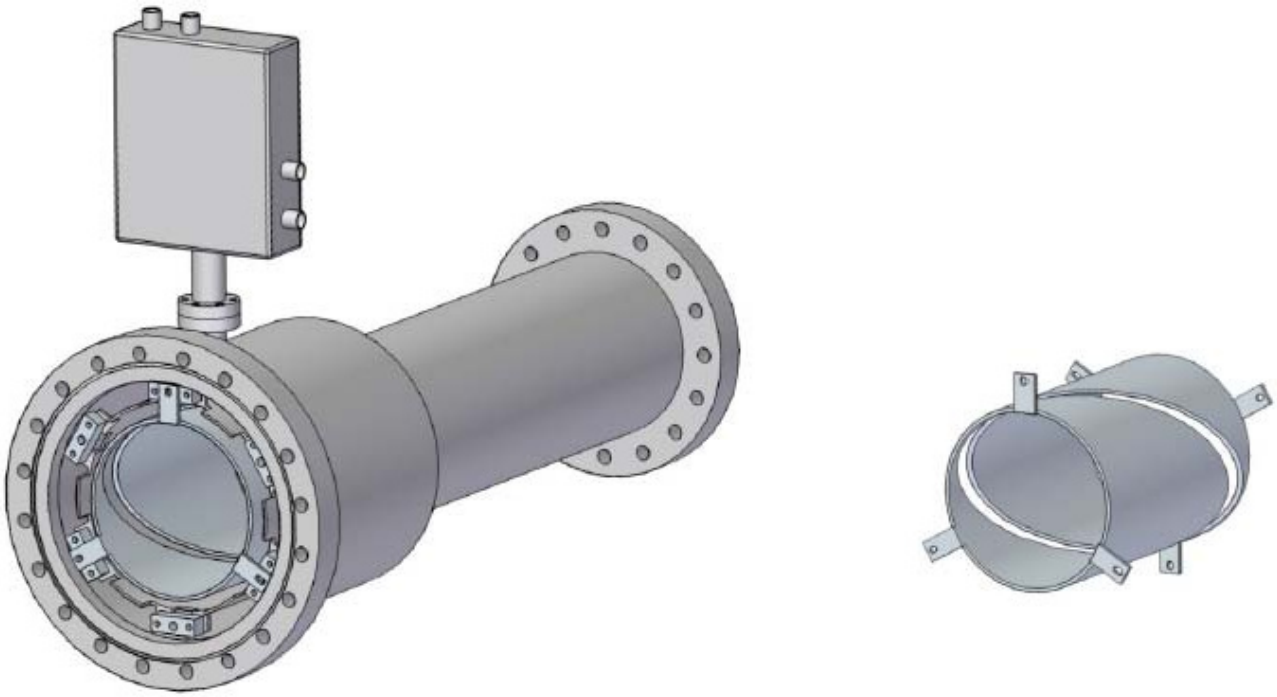

Figure 1.1: Mechanical drawing of a single BPM. The two segments of the slotted tube are the capacitor plates. The box on the top is the amplifier. Image origin: [3]

The 36 voltages of the capacitor plates are amplified and led via coaxial cables to a single evaluation point where the analog to digital conversion and the digital processing takes place.

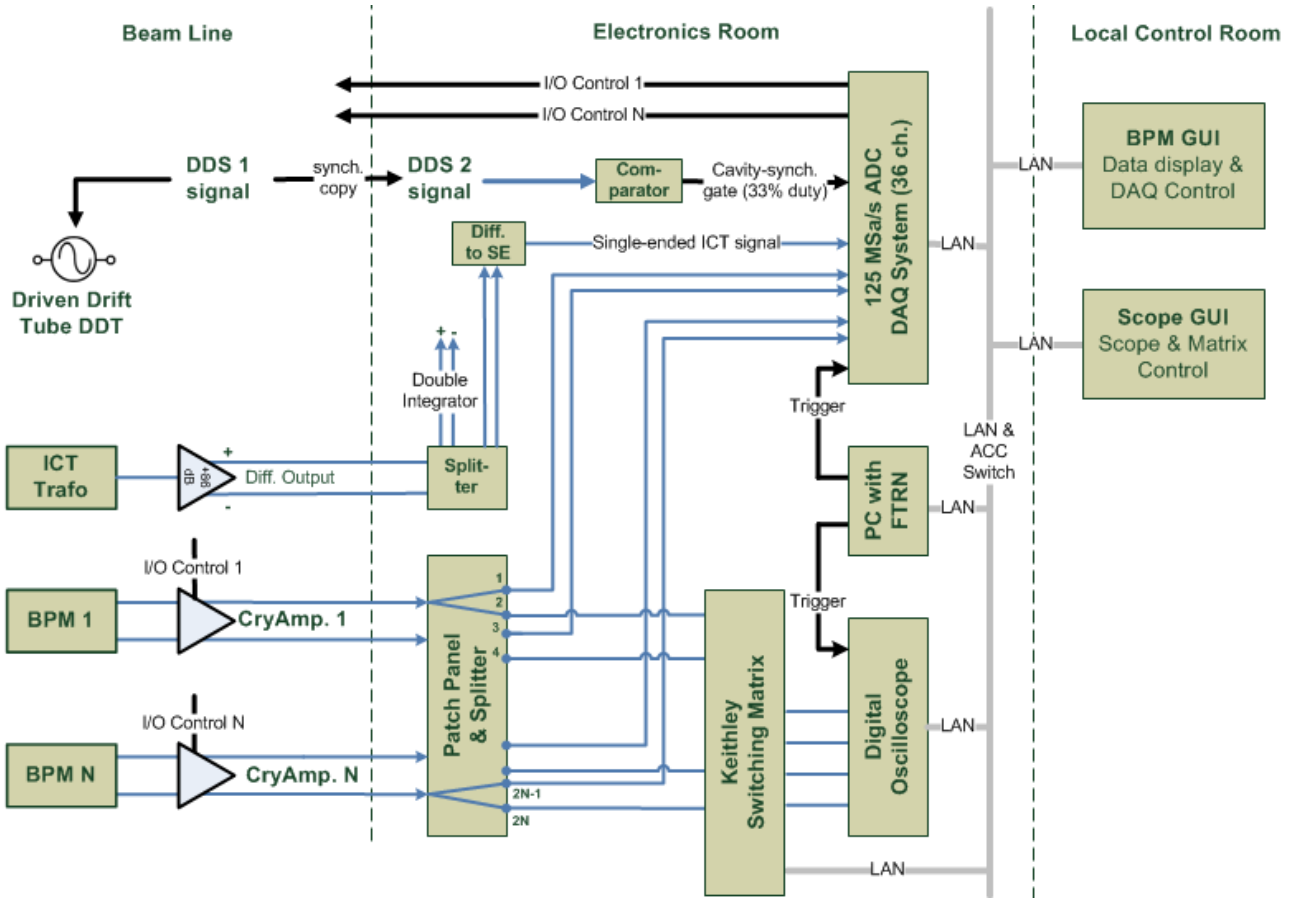

Figure 1.2: Cryring BPM system overview. This document only describes the implementation of the block labeled *125 MSa/s ADC DAQ System (36 ch.)*, excluding the software part. Image origin: [3]

The positions of the particle beam are calculated respectively from the voltage difference of two related capacitor plates using the algorithm described in chapter 2.

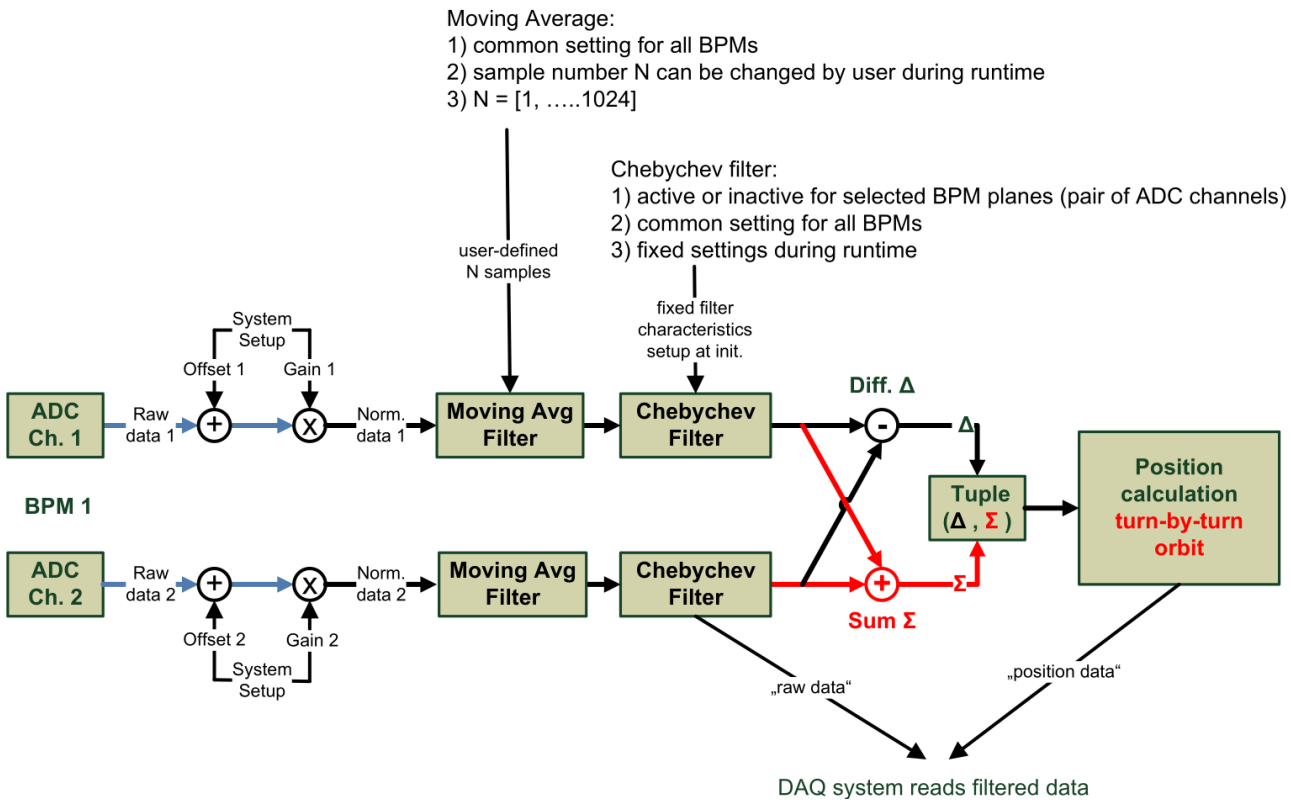

Figure 1.3: Schematic of the digital input filters. Image origin: Andreas Reiter

The moving average filter is intended to reduce the noise when higher frequencies are not of interest.

The Chebyshev filter is intended to suppress a 70 kHz noise on certain BPMs, which couples in from the supply voltage of the ion getter pumps.

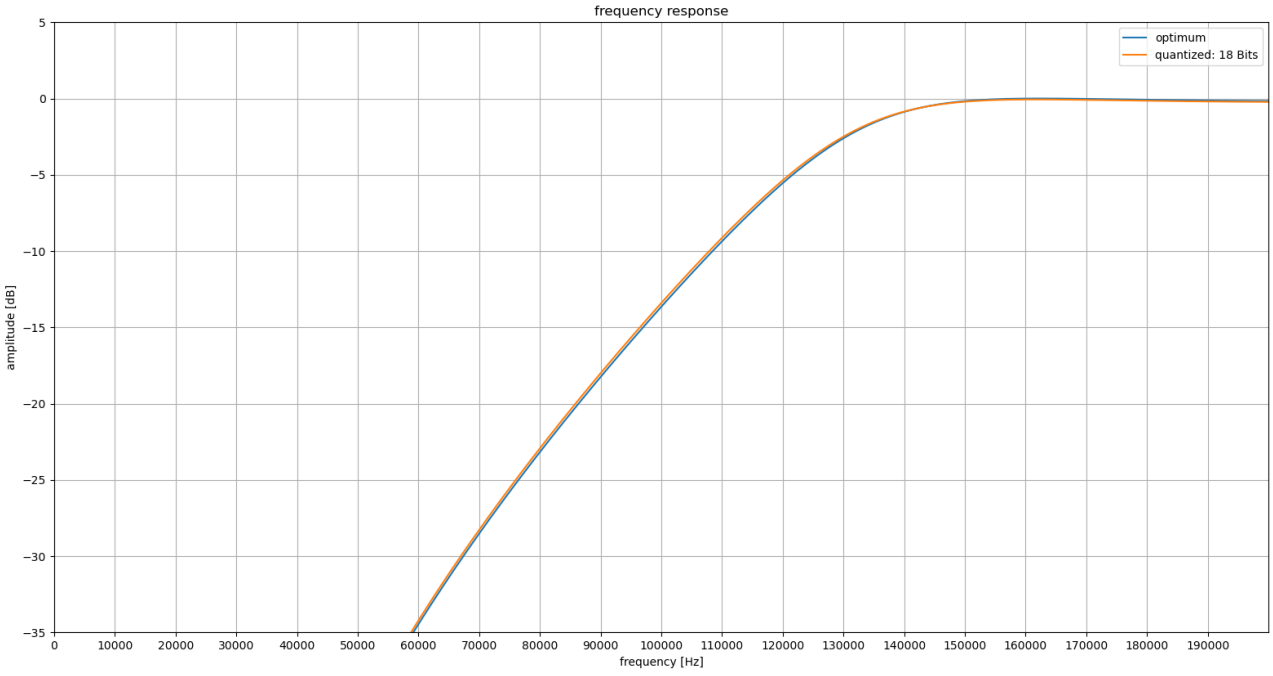

Figure 1.4: Frequency response of the Chebyshev filter

1.2 Processing hardware

Each of the 36 voltages coming from the amplifiers at the capacitor plates is sampled by a Renesas ISLA216P ADC at a sampling rate of 125 MHz with a resolution of 16 bits. Respectively four of the ADCs are placed on a single FMC board. Respectively two (ore only one for the last one) of the FMC boards are mounted on an AFC carrier board which is equipped with a Xilinx Artix XC7A200T FPGA for data processing [4].

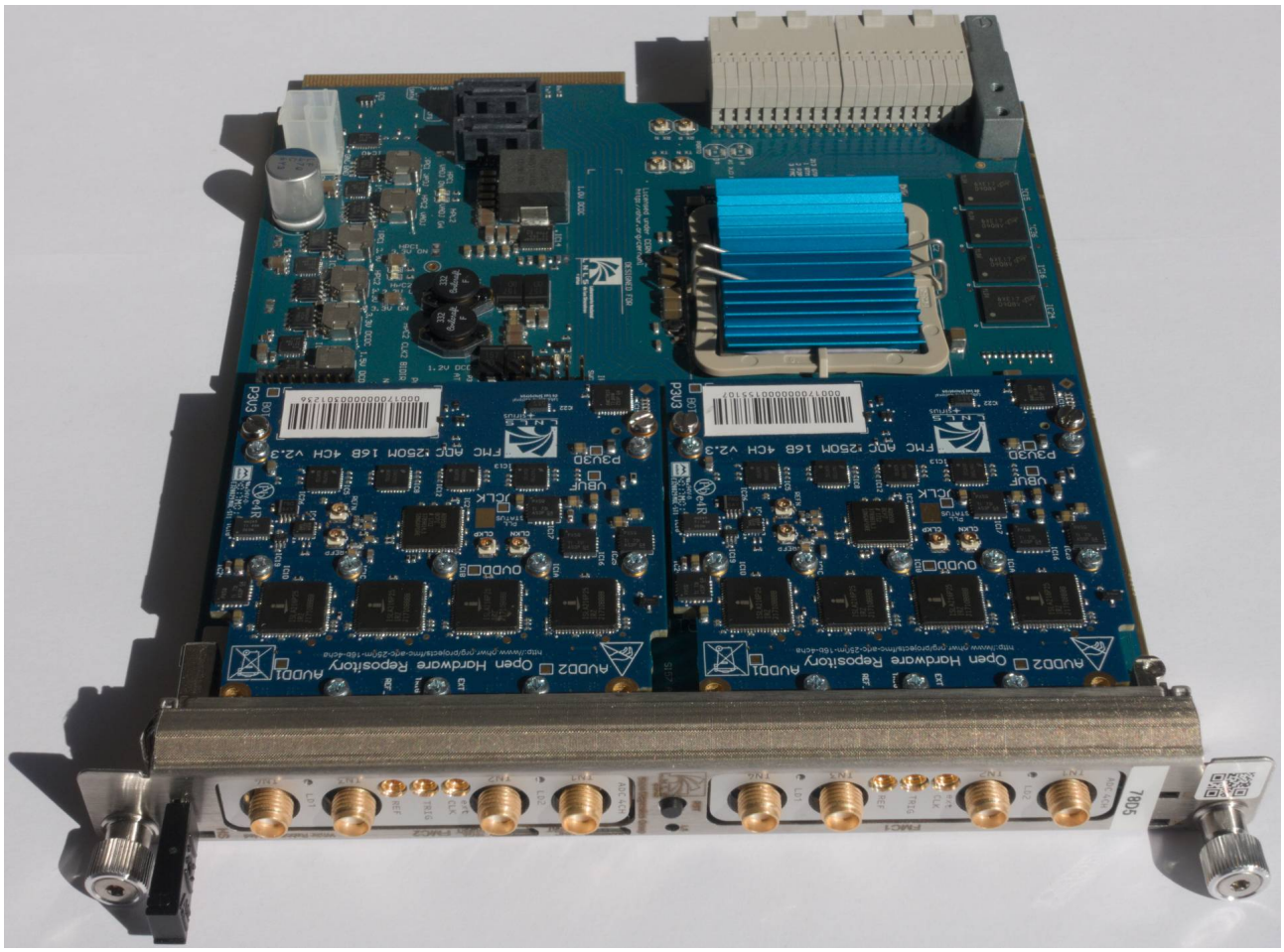

Figure 1.5: An AFC carrier board with two mounted ADC FMC boards. The FPGA is located under the blue heat spreader.

The FPGA is a mid-range device providing the following resources [5]:

- Logic cells: 215,360
- Block RAMs: 365 x 36 Kibits
- Multipliers/Adders: 740

The whole system uses five AFC carrier boards which are mounted in a MicroTCA crate together with a timing receiver and a FEC for post processing. Each of the five FPGAs is responsible for the processing of up to eight ADC data streams. The communication between the FEC and the FPGAs takes place via PCI Express over the so called backplane of the MicroTCA crate.

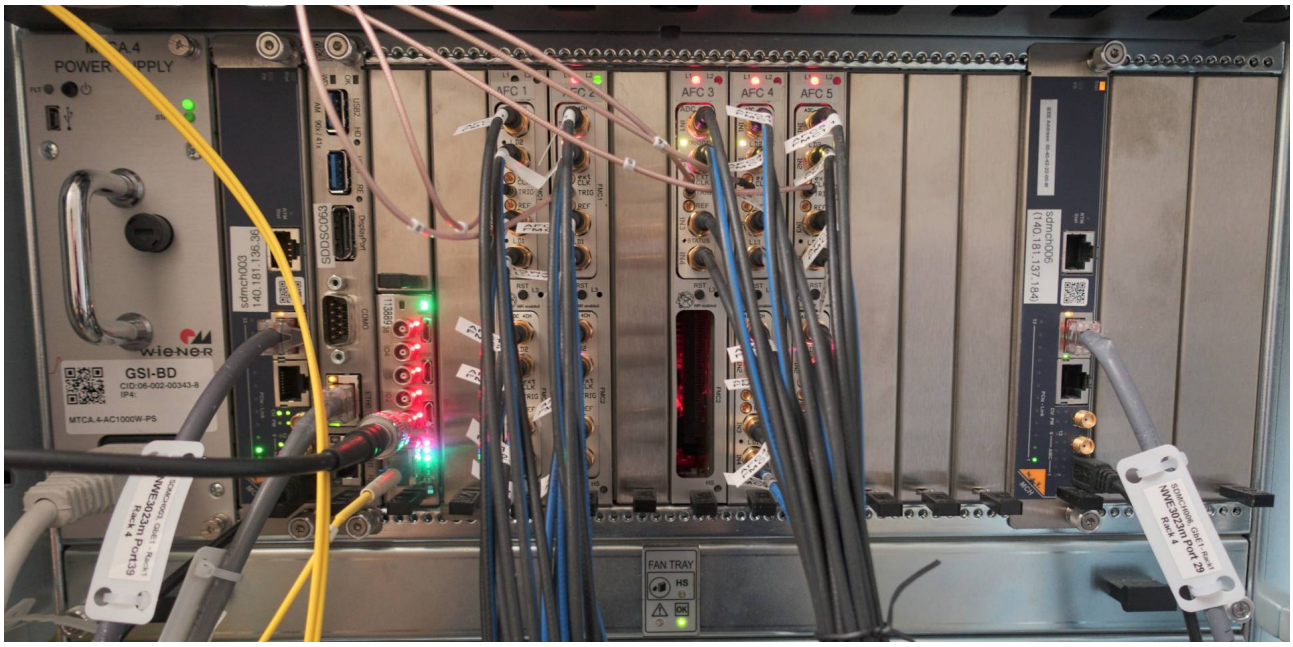

Figure 1.6: MicroTCA crate with from left to right: power supply, MCH, FEC, timing receiver, 5 AFC boards with 9 mounted FMC ADC boards, second MCH

This document describes the gatewares of the FPGAs on the five AFC carrier boards. The gatewares are identical independent of the number of mounted FMC boards.

2 BPM algorithm

The beam position is calculated from the measurement of the voltages of two corresponding plates:

$$\delta = a + \frac{x}{\kappa}\sigma \tag{2.1}$$

with

$$\delta = U_R - U_L \tag{2.2}$$

and

$$\sigma = U_R + U_L \tag{2.3}$$

where κ is a proportionality factor influenced by the dimension of the measurement system, a some possible voltage offset and x the beam position.

2.1 Capacitance correction

The capacitance of the two corresponding capacitor plates can differ from their nominal value so that one of the voltages has to be corrected by multiplying a correction factor:

$$U_R = U_{R,orig} \tag{2.4}$$

$$U_L = c_L \cdot U_{L,orig} \tag{2.5}$$

The default value of c_L in the gateware is 1. It is configurable by the software via register accesses.

2.2 Least squares algorithm

A linear least squares approach is used to reduce measurement errors. The choice of the algorithm is described in [1]. The optimal approach would be an orthogonal least squares algorithm. Since the relative error of the difference signal δ dominates that of the sum signal σ, it can be simplified to a vertical least squares algorithm:

$$E(x,a) = \sum_i (a + \frac{x}{\kappa}\sigma_i - \delta_i)^2 \tag{2.6}$$

Minimizing

$$E(x,a) \tag{2.7}$$

via partial differentiation

$$\frac{\partial E}{\partial x} = 0 \tag{2.8}$$

and

$$\frac{\partial E}{\partial a} = 0 \tag{2.9}$$

leads to

$$\frac{x}{\kappa} = \frac{N\sum_i \sigma_i\delta_i - (\sum_i \sigma_i)(\sum_i \delta_i)}{N\sum_i \sigma_i^2 - (\sum_i \sigma_i)^2} \tag{2.10}$$

2.2.1 Variance

The variance of the least squares algorithm result is calculated as follows:

$$Var = \frac{N\sum_i \delta_i^2 - (\sum_i \delta_i)^2}{N(N\sum_i \sigma_i^2 - (\sum_i \sigma_i)^2)} \tag{2.11}$$

2.2.2 Intensity

The beam intensity is calculated as follows:

$$I = \frac{N\sum_i \sigma_i^2 - (\sum_i \sigma_i)^2}{N^2} \tag{2.12}$$

2.3 Averaging

For further reducing the data rate and reducing the measurement noise, the result of the least squares algorithm is averaged over an adjustable number of samples N. This is implemented via a simple block averaging:

$$x_{avg} = \frac{1}{N}\sum_{t=0}^{N-1} x(t) \tag{2.13}$$

2.3.1 Variance

The variance of the averaging result is calculated by averaging the variances and dividing by the averaging length:

$$Var_{avg} = \frac{1}{N^2}\sum_{t=0}^{N-1} Var(t) \tag{2.14}$$

2.3.2 Intensity

The average intensity is calculated as follows:

$$I_{avg} = \frac{1}{N}\sum_{t=0}^{N-1} I(t) \tag{2.15}$$

2.4 Control signals

2.4.1 Gate

There is an input signal coming from a timing receiver that gates the calculation of the least squares algorithm. By default, the gateware is configured to use the first MLVDS line as an input for the gate signal. A calculation will start with the low to high transition of the gate signal and will be repeated continuously until a high to low transition is detected, after which the current calculation will still be completed.

2.4.2 RF pulse

The RF pulse signal is intended to synchronize the calculation of the least squares algorithm with the frequency of the particle bunches. A possible previous calculation of the least squares algorithm will be finished and a new calculation will be started whenever a RF pulse is detected.

2.5 Parameters

2.5.1 Least squares algorithm calculation length

This parameter defines the number of ADC samples that will be taken into account by the least squares algorithm if no RF pulses are present. The detection of a RF pulse will override this parameter. The overriding will only work as expected if the calculation length is set to a value that is longer than the period of the RF pulses.

The available range of values for the calculation length is 3 to 65536.

2.5.2 Averaging length

This parameter defines the number of least squares algorithm results that will be taken into account by the averaging algorithm.

The available range of values for the averaging length is 1 to 1048576.

3 Common FPGA based projects documentation

This project incorporates the code from the *FPGA_Common* Git repository which is used in multiple projects. The documentation of the common features can be found here:

3.1 Common monitoring and control features

Documentation about the register bank, the architecture information storage and the observer can be found here:
https://git.gsi.de/BEA_HDL/FPGA_Common#2-common-monitoring-and-control-features

3.2 FPGA Observer

There is a expert GUI that can be used together with multiple projects:
https://git.gsi.de/BEA_HDL/FPGA_Common#3-fpga-observer

3.3 Build flow and simulation

You can find instructions on how to build and simulate the gateware here:
https://git.gsi.de/BEA_HDL/FPGA_Common#4-build-flow-and-simulation

3.4 Helper scripts

You can find usefull scripts here:
https://git.gsi.de/BEA_HDL/FPGA_Common#5-helper-scripts

3.5 Continuous integration environment

Information about the continuous integration setup can be found here:
https://git.gsi.de/BEA_HDL/FPGA_Common#6-continuous-integration-environment

3.6 Programming and hardware configuration

You can find instructions on how to program the FPGA and configure other hardware here:

https://git.gsi.de/BEA_HDL/FPGA_Common#7-programming-and-hardware-configuration

4 Peripheral devices

There are three different peripheral devices on each of the FMC ADC boards that have to be configured by the gateware. Since they have no persistent storage they have to configured after every power cycle:

- Si571 programmable VCXO
- AD9510 PLL and clock distribution
- ISLA216P ADC

4.1 Si571 programmable VCXO

The Si571 programmable VCXO is connected via I2C using 0x49 as device address. Additionally, there is an OE (output enable) pin, which has to be driven high or left unconnected since it provides an internal pullup. The device supports a maximum I2C bus speed of 400 kbit/s.

The startup frequency before configuration via I2C is 155.52 MHz. The Si571 is located below the heat spreader of the FMC board, which has to be unscrewed to read the labeling:

*SiLabs 571
AJC000337 G
D09JW702+*

The part properties can be decoded by providing the part number 571AJC000337 on a SiLabs web page [6]:

Product: Si571
Description: Differential/single-ended I2C programmable VCXO; 10-1417 MHz
Frequency A: 155.52 MHz
I2C Address (Hex Format): 49
Format: LVPECL
Supply Voltage: 3.3 V
OE Polarity: OE active high
Temperature Stability: 20 ppm
Tuning Slope: 135 ppm/V
Minimum APR: +/- 130 ppm
Frequency Range. 10 - 280 MHz
Operating Temp Range (C): -40 to +85

A datasheet can be found on the SiLabs website [7].

4.1.1 Programming the frequency

There are three adjustable parameters that define the output frequency:

$$f_{out} = \frac{f_{XTAL} \cdot RFREQ}{HSDIV \cdot N1} \tag{4.1}$$

where

- f_{XTAL} is the fixed internal quartz frequency of 114.285 MHz +/- 2000 ppm.
- $f_{XTAL} \cdot RFREQ$ has to be in the range $[4850\text{MHz}, 5670\text{MHz}]$.
- allowed values for $HSDIV$ are 4, 5, 6, 7, 9, 11
- allowed values for $N1$ are 1 and all even numbers in $[2, 128]$

The three parameters should be chosen in a way that $RFREQ$ is minimal to reduce power consumption. If there should still be multiple possibilities for the choice of $HSDIV \cdot N1$, one should choose $HSDIV$ as maximal.

For a desired output frequency of 125 MHz the optimum values are:

- $HSDIV$ = 5
- $N1$ = 8
- $RFREQ$ = 43.750273439

Since the uncorrected f_{XTAL} frequency has an inaccuracy of 2000 ppm, one should read the initial $RFREQ$ value first and calculate

$$RFREQ = RFREQ_{init} \cdot \frac{f_{out} \cdot HSDIV \cdot N1}{f_{out,init} \cdot HSDIV_{init} \cdot N1_{init}} \tag{4.2}$$

in order to get a more accurate result. $RFREQ_{init}$ is factory calibrated to compensate the actual frequency offset of f_{XTAL}.

The VCXO has a built-in configuration timeout of 10 ms. All I2C write operations from freezing to unfreezing the digitally controlled oscillator have to complete during this period to become active.

4.1.2 Configuration

The following registers are read by the gateware for calculating the frequency correction:

address	description
0x07	HSDIV - 4 (bits 7 - 5), N1 - 1 MSB (bits 4 - 0)
0x08	RFREQ MSB (bits 5 - 0)
0x09	RFREQ
0x0A	RFREQ
0x0B	RFREQ
0x0C	RFREQ LSB

Table 4.1: Si571 registers read by the gateware

The value of register 0x07 is only used to determine if the frequency has been programmed before, e.g. after a reloading of the bitstream of the FPGA without a power cycle of the FMC ADC board. The initial value of HSDIV is 4 and it is programmed to 5. Applying the frequency correction again would lead to a wrong result, since the RFREQ registers do not contain the factory defaults any more.

The following registers are programmed by the gateware after the calculation of the frequency correction:

address	value	description
0x89	0x10	freeze digitally controlled oscillator (bit 4)
0x07	0x21	HSDIV - 4 (bits 7 - 5), N1 - 1 MSB (bits 4 - 0)
0x08	0xC2	N1 - 1 LSB (bits 7 - 6), RFREQ MSB (bit 5 - 0)
0x09	result of the frequency correction	RFREQ
0x0A	result of the frequency correction	RFREQ
0x0B	result of the frequency correction	RFREQ
0x0C	result of the frequency correction	RFREQ LSB
0x89	0x00	unfreeze digitally controlled oscillator (bit 4)
0x87	0x40	new frequency applied (bit 6)

Table 4.2: Si571 registers programmed by the gateware

Example configuration of some devices

FMC version	FMC SN	initial RFREQ	programmed RFREQ	meas. freq. (Hz)
v2.3	155107	0x02B8EF1D6D	0x02BC3497C2	125024480
v2.3	301236	0x02B8F28CD4	0x02BC380B4A	125024401
v1.2	-	0x02B8BC2738	0x02BC016450	124974845
v1.0	-	0x02B94432E0	0x02BC8A1373	124975700

The frequency was measured relatively to the processing clock of the AFC board.

4.2 AD9510 PLL and clock distribution

The Analog Devices AD9510 is connected via SPI. Writing to registers must be completed with a write to the register address 0x5A with the LSBit set in the write value (e.g. 0x01) to take effect. Multiple writes can precede the writing of register 0x5A, so that this needs to be done only once at the end of a write sequence. The maximum SPI clock frequency is 25 MHz.

The phase frequency detector of the PLL, which compares the VCXO frequency to the reference frequency, has a maximum input frequency of 100 MHz. Higher frequencies have to be divided by the prescalers R (reference input) and N (VCXO input). A lock signal can be connected to a status pin, that is connected to a FPGA GPIO.

A datasheet can be found on the Analog Devices website [8].

4.2.1 Configuration

The gateware configures the AD9510 device to lock the VCXO frequency to a reference clock coming from the FPGA. The following registers are programmed by the gateware:

address	value	description
0x08	0x33	normal charge pump mode (bits 1 - 0), analog lock detect on STATUS pin (bit 5 - 2)
0x09	0x70	charge pump current: 4.8 mA (bits 6 - 4)
0x0A	0x44	PLL power up (bits 1 - 0), VCXO prescaler: 2 (bits 4 - 2), B counter bypass (bit 6)
0x0B	0x00	R divider MSB
0x0C	0x02	R divider LSB: divide reference clock input by 2
0x0D	0x02	anti backlash pulse width: 6.0 ns (bit 1 - 0)
0x3C	0x08	output 0 voltage: 810 mV (bits 3 - 2), output 0 enable (to ADC 0, bits 1 - 0)
0x3D	0x08	output 1 voltage: 810 mV (bits 3 - 2), output 1 enable (to ADC 1, bits 1 - 0)
0x3E	0x08	output 2 voltage: 810 mV (bits 3 - 2), output 2 enable (to ADC 2, bits 1 - 0)
0x3F	0x08	output 3 voltage: 810 mV (bits 3 - 2), output 3 enable (to ADC 3, bits 1 - 0)
0x40	0x03	output 4 power down (bit 0)
0x41	0x03	output 5 power down (bit 0)
0x42	0x03	output 6 power down (bit 0)
0x43	0x02	output 7 current 3.5 mA (bits 2 - 1), output 7 enable (to FPGA for monitoring, bit 0)
0x45	0x02	clk1 power down (bit 1), clock input from clk2 (VCXO) (bit 0)
0x49	0x80	bypass the divider in front of output 0 (bit 7)
0x4B	0x80	bypass the divider in front of output 1 (bit 7)
0x4D	0x80	bypass the divider in front of output 2 (bit 7)
0x4F	0x80	bypass the divider in front of output 3 (bit 7)
0x57	0x80	bypass the divider in front of output 7 (bit 7)
0x5A	0x01	load the register bank overlay to the actual register bank (bit 0)

Table 4.3: AD9510 registers programmed by the gateware

4.3 ISLA216P ADC

The four ISLA216P ADCs are connected via SPI. The communication to each chip is enabled via an individual chip select line. The MOSI, MISO and CLK lines are shared between the four chips. Parallel configuration by driving all chip selects high at the same time works for the writing registers, but not for reading, since there would be multiple drivers on the MISO line. The maximum SPI clock frequency is given by the ADC sampling frequency divided by 16. At a sample frequency of 125 MHz this corresponds to a SPI clock frequency of 7.8125 MHz.

A datasheet can be found on the Renesas website [9].

The ADCs provide a configurable gain correction of +/- 4.2Since the gain correction and the offset correction are implemented digitally in the gateware, most of the configuration registers can be left at their default values.

The SPI interface in the gateware is implemented as a four wire interface, whereas the default setting of the ISLA216P SPI interface is a three wire mode. For being able to configure the ISLA216Ps interactively, the gateware configures the corresponding register to four wire mode at start up.

4.3.1 Configuration

The following registers are programmed by the gateware:

address	value	description
0x00	0x80	enable four wire mode (enable the usage of a dedicated SPI MISO line)

Table 4.4: ISLA216P registers programmed by the gateware

5 Gateware implementation

5.1 Clocking

The gateware uses three primary clocks:

- PCIe reference clock, 100 MHz
- FMC 0 ADC clock, 125 MHz
- FMC 1 ADC clock, 125 MHz

5.1.1 PCIe reference clock

The PCIe reference clock comes from an output of an ADN4604 clock switch on the AFC board [11]. The clock switch is controlled via I2C by the MMC firmware to output a 100 MHz clock, which enters the FPGA as a differential input signal on pins H20 and G20. The PCIe reference clock feeds the reference clock input of the PCIe IP core by Xilinx, which contains a PLL producing a 125 MHz output clock for the AXI interface named *clk_125_pcie_axi*.

clk_125_pcie_axi drives a MMCM to generate:

- *clk_125*, 125 MHz, this is the main processing clock of the design
- *clk_100*, 100 MHz, used for reading the FPGA serial number
- *clk_200*, 200 MHz, used for the SDRAM interface IP core by Xilinx

The SDRAM interface IP core contains a MMCM which generates a 100 MHz clock named *clk_100_sdram* for the AXI interface.

5.1.2 FMC ADC clocks

On each of the two FMC boards there is a Si571 programmable VCXO (see chapter 4.1) which feeds the four ADCs. The frequency of the VCXO can be coupled to a reference clock coming from the FPGA (see chapter 4.2). The VCXO is programmed to a nominal output frequency of 125 MHz and coupled by the PLL with *clk_125* coming from the FPGA. Bringing the PLL to lock is quite demanding, so that with the current settings a stable lock can not be guaranteed.

From each of the four ADCs of an FMC board an individual clock signal is led to the FPGA which is used for the de-serialization in the IDDR primitives. For the further processing, only the clock signal from the first ADC of a FMC board is used since the clock frequencies of the four ADCs are identical.

There are two clock domain crossing FIFOs in the gateware to synchronize the data from the ADCs to the main processing clock *clk_125*.

5.2 Resets

5.2.1 PLL not in lock

As long as the PLL in the MMCM producing the main processing clock *clk_125* is not yet in lock, the design is held in reset. After this the lock should be stable until the next power cycle.

5.2.2 PCIe reset

The design will be reset whenever the PCIe connection is re-initialized. This happens e.g. when the FEC is rebooted.

5.2.3 Reset button

There is a push button labeled *RST* at the center of the AFC front panel which is connected to the microcontroller for the MMC firmware. The OpenMMC firmware forwards a button press with a duration of at least two seconds to the FPGA pin AG26 as an active low signal to initiate a reset of the gateware.

Resetting the FPGA leads to the loss of the PCIe connection. To re-enable the connection, the FEC has to be rebooted.

5.3 Data flow diagram

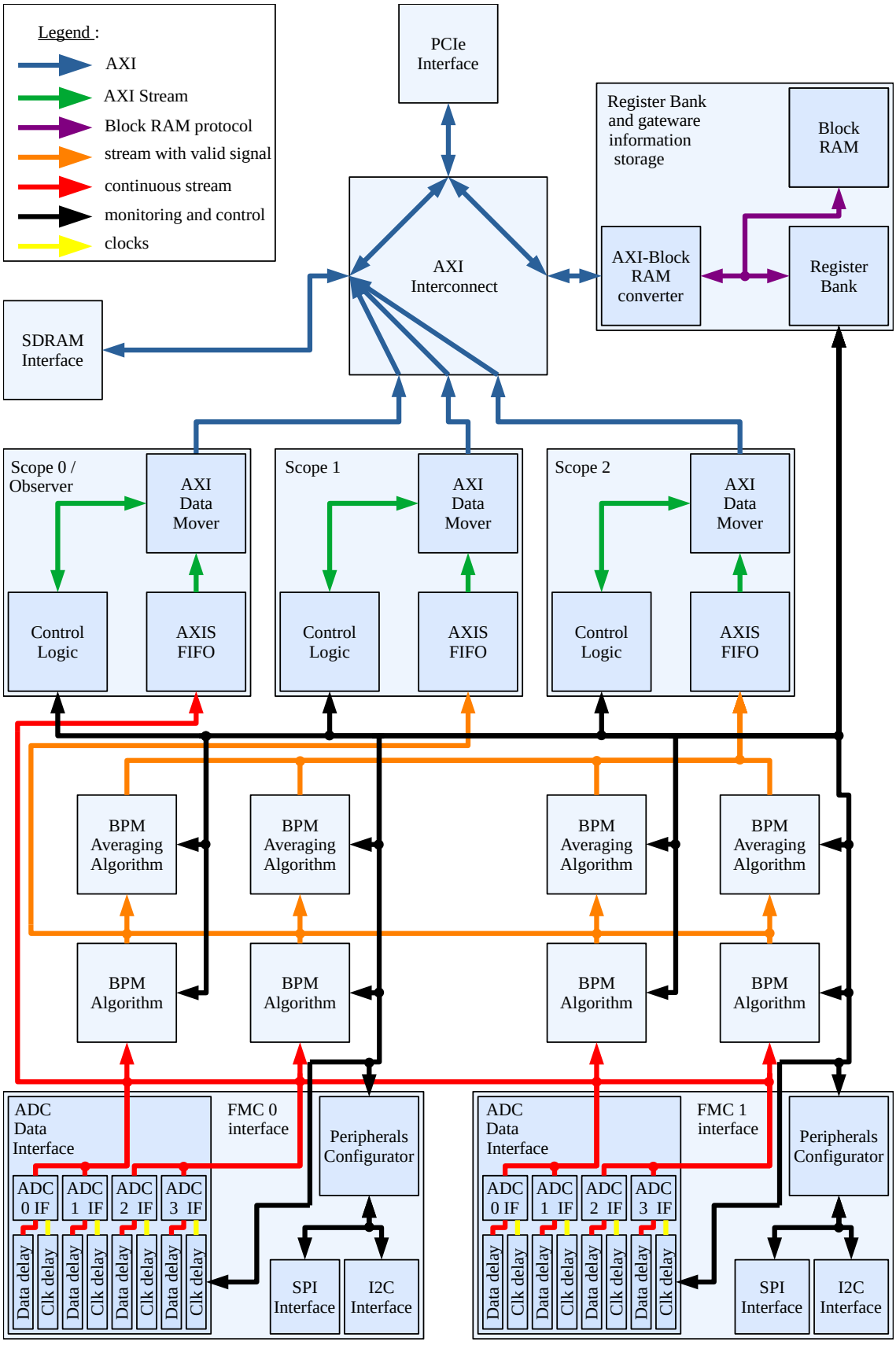

Figure 5.1: Simplified data flow diagram

Figure 5.1 shows a simplified data flow diagram. For simplicity, some features are not included in the diagram:

- processing clocks and clock domain crossings
- resets
- gate and RF pulse inputs
- ADC data offset and gain corrections
- LEDs
- read out logics of FPGA serial number and build time stamp
- ADC maximum amplitude calculation
- signals connected to the observer
- data width conversions of AXI and AXI Stream connections

5.4 Input delays

The data inputs from the ADCs require a latency correction to compensate clock and routing delays. This is implemented via individually configurable input delay primitives for both the clock and for the data input pins. By increasing the input delay of either a clock or of the associated data inputs, the alignment can be corrected in both directions.

The input delays provide a 32 tap delay line with a configurable delay between 0 to 31 taps [10]. Each tap corresponds to a delay of:

$t_{tab\ delay} = \frac{1}{64 \cdot f_{ref}}$

with f_{ref} being the frequency of the clock connected to the IDELAYCTRL primitive. With the 200 MHz clock connected in this design this corresponds to a tap delay value of 78 s.

5.4.1 Calculation of optimal delay values

The ADCs offers programmable user patterns that can be sent in place of the ADC samples to check the correct timing of the digital interface. For finding the optimum delay values, the following procedure is applied:

- The clock input delay is increased until the pattern begins to deteriorate and the delay index at which that happens is noted.
- After that the clock input delay is reset to 0 and the data delay is increased until the pattern begins to deteriorate.
- The optimum value is assumed to be the midpoint between these values.

5.4.2 Chosen delay values

For an ADC clock frequency of 125 MHz the optimal delay values are:

- FMC0: no deterioration at any clock delay, but deterioration at and above data delay 0x06
  - ADC clock delay value: 0x0D
  - ADC data delay value: 0x00
- FMC1: no deterioration at any clock delay, but deterioration at and above data delay 0x05
  - ADC clock delay value: 0x0D
  - ADC data delay value: 0x00

These values are programmed to the IDELAY primitives at start up. They can be changed via individual configuration registers (see chapter 7.1.2).

5.5 Clock domain crossings

Even though the FMC clocks are coupled to the main processing clock by the FMC's PLLs, they can jitter against the main processing clock or even run at a slightly different frequency if the PLLs unlock for any reason.

To prevent data corruption, two clock domain crossing FIFOs are used for the incoming ADC data, one for each FMC board. In the case of frequency deviations, there are two cases to differentiate:

1. the FMC clock is running slightly faster than the processing clock:
   - one sample at a time will be discarded
   - this happens synchronous for all four ADCs of a FMC board
2. the FMC clock is running slightly slower than the processing clock:
   - one sample at a time will be repeated
   - this happens synchronous for all four ADCs of a FMC board

Due to the synchronous handling of the four ADCs on a FMC board, the discarding or repetition of samples should not have any measurable effect on the BPM results, since the two inputs to each BPM come from the same FMC board.

5.6 Gate signal

The gate signal is fed to the FPGA via one of the eight MLVDS lines on the AMC connector. The selection is made via a configuration register (see chapter 6.2.2). The register's default is to route MLVDS line 0 to the gate input of the BPM algorithm.

5.7 RF signal

The pulses of the RF signal define the length of the linear regression in the BPM algorithm. The selection is made via a configuration register (see chapter 6.2.2). As a default, the RF signal is fed to the FPGA via the MMCX connector labeled *TRIG* on the front panel of FMC 1. In this case this signal is valid for both FMCs and the *TRIG* input of FMC 2 is not used.

5.8 BPM algorithm

The proportionality factor κ in equation 2.10 is set implicitly to 1 so that the result has to interpreted as a relative position in the range $[-1, 1]$:

$$x = \frac{N \sum_i \sigma_i \delta_i - (\sum_i \sigma_i)(\sum_i \delta_i)}{N \sum_i \sigma_i^2 - (\sum_i \sigma_i)^2} \tag{5.1}$$

The capacitance correction (see chapter 2.1) and the linear regression (equation 4.1) are implemented as a pipelined algorithm. The processing clock is equal to the sampling frequency of the ADCs.

5.8.1 Pipeline steps performed every clock cycle

The gain correction, the differences and sums of the incoming ADC data pairs σ and δ and the four different sums of equation 4.1 are calculated every clock cycle.

Step 0: Capacitance correction

- offset and gain corrected ADC 0 data sample is strobed unchanged. Input: 17 bits signed, output 17 bits signed
- offset and gain corrected ADC 1 data sample is multiplied with a correction factor coming from a configuration register. Input: 17 bits signed for data, 16 bits unsigned for correction factor, output 17 bits signed

Step 1: Calculation of sum and difference signals

- σ: sum of data 0 and data 1, inputs: 17 bits signed, output: 18 bits signed
- δ: difference of data 0 and data 1, inputs: 17 bits signed, output: 18 bits signed

Step 2: Calculation of products, sign extension, summation

The maximum word length of the adders in the DSP48 blocks of the FPGA is 48 bits. When using these adders, the maximum summation length is limited by the word length of longest term $\sigma\delta$ (36) to a value of 12.

- $\sum_i \sigma_i\delta_i$: inputs: 18 bits signed, output: 48 bits signed
- $\sum_i \sigma_i^2$: input: 18 bits signed, output: 48 bits signed
- $\sum_i \sigma_i$: input: 18 bits signed, output: 30 bits signed
- $\sum_i \delta_i$: input: 18 bits signed, output: 30 bits signed
- N: counter, output: 12 bits unsigned

5.8.2 Pipeline steps performed with a reduced data rate

The following pipeline steps are only performed once for every linear regression period. The length of the linear regression period is defined by the BPM linear regression length register (see chapter 6.2.2).

If a RF signal is present, the length is additionally controlled by the distances of the pulses of this signal. A new linear regression calculation will be started with every rising edge of the RF signal, while the post processing steps for the previous period will be started.

Step 3: Conversion to floating point

The DSP48 blocks in the FPGA can only handle multiplications up to 18 bits times 25 bits. For this reason, a conversion to a floating point format is performed.

The floating point format is:

- $mantissa$: 18 bits signed (integer, not fractional as usual for floating point formats)
- $exponent$: 6 bits unsigned

which decodes to: $value = mantissa \cdot 2^{exponent}$

The sums $\sum_i \sigma_i\delta_i$, $\sum_i \sigma_i^2$, $\sum_i \sigma_i$ and $\sum_i \delta_i$ are converted to float.

A conversion is not necessary for N since it is only 12 bits wide.

Step 4: Calculation of the products of sums

The products $N\sum_i \sigma_i\delta_i$, $(\sum_i \sigma_i)(\sum_i \delta_i)$, $N\sum_i \sigma_i^2$ and $(\sum_i \sigma_i)^2$ are calculated by multiplying the mantissas and by adding the exponents of the floating point representations.

Step 5: Shifting to align for subtraction and sign extensions

In general the results of step 4 will have different exponents, so that the mantissas have to be shifted to a common exponent before a subtraction can take place.

The mantissa of the float number with the smaller exponent is shifted by the difference of exponents digits to the right and the exponent is set to the larger exponent. Sign extensions by 1 bit take place to prevent over- and underflows by the subtraction.

Step 6: Calculation of the subtractions in the numerator and the denominator

Now that the operands have the same exponent, the subtractions can take place by subtracting the mantissas.

The exponents of the results stay the same as that of the operands.

The results are: $N \sum_i \sigma_i \delta_i - (\sum_i \sigma_i)(\sum_i \delta_i)$ and $N \sum_i \sigma_i^2 - (\sum_i \sigma_i)^2$

Step 7: Conversion of the mantissas to floating point

Due to the multiplication in step 4 and the sign extension in step 5 the mantissas have now a length of 37 bits, which is again too long for the final division. The mantissa is converted to the same floating point format as described in step 4.

The results respectively have a mantissa of 18 bits and two exponents of 6 bits each which have to be united in the next step.

Step 8: Unification of exponents and start of division

Division is a costly operation in FPGAs. In this implementation it is performed by an IP core by Xilinx which is parametrized to 18 bits for both the divisor and the dividend. The result is 33 bits wide, of which 15 bits are fractional.

The division takes 25 clock cycles to complete. The divider IP core reaches a throughput of 1 in 3 clock cycles. Thus 3 is the lower limit for the linear regression length for the current settings of the IP core.

The exponents generated in step 7 are united to the existing ones from step 6 by addition.

Step 9 - 32: Waiting for the division to complete

The results of step 8 are pipelined until the completion of the division.

Step 33: Shifting and slicing the division result

The division result is shifted to the right by minus the exponent from step 8. After that, the lower 16 bits are sliced to form the result of the linear regression algorithm.

The result has to be interpreted as a relative position in the range $[-1, 1[$, multiplied by 2^{15}.

Two signals are created for debugging purposes and are connected to the observer (see https://git.gsi.de/BEA_HDL/FPGA_Common#23-observer):

- *result out of range* (1 bit): High if the absolute value of the numerator is greater than that of the denominator. This can happen if the phases of the two input signals are not aligned. In this case the result is set to the maximum or minimum value.
- *division by zero* (1 bit): Comes from the divider IP core and is high if the divisor is zero. This is very unlikely to happen. In this case the result is set to 0.

Limitations

Allowed values for the linear regression length are: 3, 4, 5, ... , 4096

The lower limit is caused by the divider IP core which can only handle one division in three clock cycles.

The upper limit is caused by the maximum operand length of the adder in the DSP48 primitives in the FPGA. A higher limit would be implementable at the cost of an increased resource usage and two additional clock cycles of processing latency.

5.9 BPM averaging

The result from the BPM algorithm is sign extended and added up until the desired number of samples is reached. Only powers of two are allowed for the averaging length. Allowing any desired number would require a general division operation at the end of the averaging process, whereas a division by a power of two can be implemented by a simple shift operation. This is why the configuration register 'log2 of BPM averaging length' contains the dual logarithm of the averaging length (see chapter 6.2.2).

The result is sliced to the same number of bits as the result from the BPM algorithm. It also has to be interpreted as a relative position in the range $[-1, 1[$, multiplied by 2^{15}.

Available values for the averaging length are 1, 2, 4, ... , 1,048,576.

The upper limit is not caused by any implementation limitation, but was simply chosen because longer averaging lengths were not assumed to be useful.

5.10 AXI infrastructure

The memory mapped data transfers inside the FPGA are handled via the AXI protocol using a star topology with a central interconnect. The common data width is 256 bits and the common clock is the main processing clock of 125 MHz.

The AXI masters connected to the interconnect are:

- PCIe interface
- scope 0
- scope 1
- scope 2

The PCIe interface only supports an AXI data width of 128 bits, so that an AXI data width converter is used to be able to connect it to the interconnect. A clock domain crossing also takes place despite identical frequencies, since the AXI clock of the PCIe interface is derived directly from the PCIe reference clock and could jitter against the independently derived main processing clock.

The AXI slaves connected to the interconnect are:

- SDRAM interface
- register bank / Block RAM

The SDRAM interface only supports an AXI clock frequency of 100 MHz, so that an AXI clock converter is used to synchronize it to the main processing clock.

The AXI interconnect is configured to connect the scopes only to the SDRAM interface and only with write access since other accesses are not needed.

Even though there is no need for the PCIe interface to write to the SDRAM, this access is enabled because otherwise the PCIe driver will crash in case of an erroneous write access to the SDRAM.

5.11 AXI Stream infrastructure

The scopes internally use an AXI Stream bus to process the incoming data. The final data stream is converted to the AXI protocol.

5.11.1 Scopes

There are three so called scopes for interactively storing calculation results. For the storage format of the scope data see chapter 6.1.

Scope 0: corrected ADC data

Since the frequency of the incoming ADC data samples is identical to the AXI clock, the data samples are parallized twice to allow flow controlled data processing. This also converts the 128 bits wide ADC data stream (8 * 16 bits) to the common AXI data width of 256 bits.

An IP core called *AXI data mover* manages the write access to the SDRAM. The block size of the AXI bus accesses is set to 4 MiBytes to allow a low protocol overhead. A block size of 256 bits (width of a single data word) would slow down the transmission in a way that the necessary data rate to store all incoming ADC data samples would not be reached.

Scope 1: BPM results

The data rate of the BPM results is slow enough so that they do not have to be parallized before the transmission.

For the same reason, the block size of the AXI bus accesses can be set to the width of a single data word which simplifies the transmission handling.

Scope 2: BPM averaging results

The data rate of the BPM averaging results is even slower than that of the BPM results, so that the same mechanism can be used.

5.12 Configuration of peripheral devices

The peripheral devices documented in chapter 4 are initially programmed by the gateware. During operation, they can be configured using the corresponding gateware registers (see chapter 7.1.2).

5.12.1 SPI Interface

There is an individual SPI interface for each of the two FMC boards. It is implemented as a four wire interface and connects to the four ADCs and to the PLL on the FMC ADC boards. The choice of the communication partner is implemented via indivial chip select lines.

5.12.2 I2C Interface

There is an individual I2C interface for each of the two FMC boards. It only connects to the VCXO. The VCXO's transaction timeout has to be kept in mind when programming it interactively (see chapter 4.1.

5.13 PCIe Interface

This gateware uses the Xilinx IP core *DMA/Bridge Subsystem for PCI Express* with the following configuration:

- PCIe speed: 5 GTransfers/s
- AXI clock frequency: 125 MHz
- reference clock frequency: 100 MHz

The PCIe reference clock is routed to the FPGA via the MMC firmware and is configured to be driven by the 100 MHz *FCLKA* clock coming from the AMC connector.

5.14 SDRAM interface

For the communication with the SDRAM, an IP core by Xilinx is used. The clock frequency of the SDRAM interface's AXI bus is 100 MHz, so that an AXI clock converter is used to connect it to the rest of the AXI infrastructure which is clocked at 125 MHz.

5.15 Observer

This project incorporates the observer interface from the *FPGA_Common* Git submodule (see https://git.gsi.de/BEA_HDL/FPGA_Common#13-observer).

The following signals are connected to the observer inputs:

value	input vector(64 bits)	valid signal
0	corrected ADC data of ADCs 0 - 3	1
1	corrected ADC data of ADCs 4 - 7	1
2	BPM 0 and 1 result, additional information	1
3	BPM 2 and 3 result, additional information	1
4	BPM 0 and 1 averaging result, additional information	1
5	BPM 2 and 3 averaging result, additional information	1
6	SPI and I2C signals, MLVDS signals, FMC trigger signals	1
7	test counter	1

The rest of the multiplexer inputs are connected to zero.

6 Gateware software interface

The communication between the gateware and the software takes place via a PCIe driver by Xilinx called XDMA. There is only one PCIe Bar in use in the gateware which maps the memory space to different physical memories on the AMC board.

The following mapping is applied:

address	size	memory type	description
0x00000000	2 kiB	Flip Flops	inside FPGA, for registers
0x00004000	16 kiB	Block RAM	inside FPGA, for architecture information
0x00010000	64 kiB	Block RAM	inside FPGA, for observer
0x80000000	2 GiB	SDRAM	external, for scope data

Table 6.1: Memory mapping

The *architecture information* and the *observer* are documented here:
https://git.gsi.de/BEA_HDL/FPGA_Common#2-common-monitoring-and-control-features

6.1 PCIe Driver

Read and write accesses are mapped to virtual file accesses:

- `/dev/xdma0_c2h_0` for read accesses
- `/dev/xdma0_h2c_0` for write accesses

6.1.1 Reading from a register

Example in C:

```
uint32_t address = 0x00000000;
int fd = open("/dev/xdma0_c2h_0", O_RDWR);
lseek(fd, address, SEEK_SET);
uint64_t value;
read(fd, &value, sizeof(uint64_t));
```

6.1.2 Writing to a register

It is important to write the whole register width of 64 bits. If a register has less than 64 bits, the unused MSBs have to be written to any value. 32 bit write accesses will not have any effect.

Example in C:

```
uint64_t value = 42;
uint32_t address = 0x00000400;
int fd = open("/dev/xdma0_h2c_0", O_RDWR);
lseek(fd, address, SEEK_SET);
write(fd, &value, sizeof(uint64_t));
```

6.1.3 Reading of scope data

Example in C:

```
uint32_t address = 0x80000000;
int fd = open("/dev/xdma0_c2h_0", O_RDWR);
lseek(fd, address, SEEK_SET);
char data[1024];
read(fd, data, 1024);
```

This example reads 1024 bytes of data from scope 0 to an array. For bigger data blocks, instead of using an array, you will probably prefer a dynamically allocated memory region.

6.2 Scope memory

There are three scope memory regions of which the one for the corrected ADC data is the largest since it has the highest data rate.

start address	size	description
0x80000000	1 GiB	corrected ADC data
0xC0000000	512 MiB	BPM result
0xE0000000	512 MiB	BPM averaging result

Table 6.2: Scopes memory map

6.2.1 Scope 0: corrected ADC data

The corrected ADC data is stored in the following format:

address	bits	radix	description
0x80000000	16	signed	ADC 0 data (time = 0)
0x80000002	16	signed	ADC 1 data (time = 0)
0x80000004	16	signed	ADC 2 data (time = 0)
0x80000006	16	signed	ADC 3 data (time = 0)
0x80000008	16	signed	ADC 4 data (time = 0)
0x8000000A	16	signed	ADC 5 data (time = 0)
0x8000000C	16	signed	ADC 6 data (time = 0)
0x8000000E	16	signed	ADC 7 data (time = 0)
0x80000010	16	signed	ADC 0 data (time = 1)
...	...	...	...

Table 6.3: Corrected ADC data storage format

The corrected data is the result of four sequential operations on the raw ADC data:

1. offset correction by adding a correction summand
2. gain correction by multiplying a correction factor
3. configurable moving average filtering
4. optional high pass filtering

The correction summand, the correction factor and the filter settings can be set by individual configuration registers (see chapter 6.2.2).

The corrected ADC data scope memory can hold up to 2^{26} samples. At a sampling frequency of 125 MHz this corresponds to a maximum capture duration of 0.537 seconds.

6.2.2 Scope 1: BPM result

The BPM result is stored in the following format:

address	bits	radix	description
0xC0000000	48	unsigned	time stamp, starting from gate high transition, 125 MHz (time = 0)
0xC0000006	16	unsigned	effective linear regression length
0xC0000008	16	signed	BPM 0 result (time = 0)
0xC000000A	16	unsigned	BPM 0 variance * N (time = 0)
0xC000000C	16	unsigned	BPM 0 intensity (time = 0)
0xC000000E	16	signed	BPM 1 result (time = 0)
0xC0000010	16	unsigned	BPM 1 variance * N (time = 0)
0xC0000012	16	unsigned	BPM 1 intensity (time = 0)
0xC0000014	16	signed	BPM 2 result (time = 0)
0xC0000016	16	unsigned	BPM 2 variance * N (time = 0)
0xC0000018	16	unsigned	BPM 2 intensity (time = 0)
0xC000001A	16	signed	BPM 3 result (time = 0)
0xC000001C	16	unsigned	BPM 3 variance * N (time = 0)
0xC000001E	16	unsigned	BPM 3 intensity (time = 0)
0xC0000020	48	unsigned	time stamp, starting from gate high transition, 125 MHz (time = 1)
...	...	...	...

Table 6.4: BPM result storage format

The BPM result scope memory can hold up to 2^{24} samples. At a sampling frequency of 125 MHz and with a linear regression length of e.g. 1024 this corresponds to a maximum capture duration of 2:17 minutes.

BPM {0 - 3} result

This value divided by 2^{15} represents the relative beam position in the range $[-1, 1[$.

BPM {0 - 3} variance * N

This value divided by 2^{16} represents the variance of the corresponding BPM result multiplied with the linear regression length.

The scaling with the linear regression length guarantees enough LSBs to evaluate. The variance itself quantized with 16 bits would otherwise often result in zero.

BPM {0 - 3} intensity

The intensity of the beam. A value of $2^{16} - 1$ corresponds to the maximum achievable intensity at an alternating pattern of maximum and minimum ADC samples on both inputs.

6.2.3 Scope 2: BPM averaging result

The BPM averaging result is stored in the following format:

address	bits	radix	description
0xE0000000	48	unsigned	time stamp, starting from gate high transition, 125 MHz (time = 0)
0xE0000006	16	unsigned	average linear regression length
0xE0000008	16	signed	BPM 0 averaging result (time = 0)
0xE000000A	16	unsigned	BPM 0 averaging variance * N * N_avg (time = 0)
0xE000000C	16	unsigned	BPM 0 averaging intensity (time = 0)
0xE000000E	16	signed	BPM 1 averaging result (time = 0)
0xE0000010	16	unsigned	BPM 1 averaging variance * N * N_avg (time = 0)
0xE0000012	16	unsigned	BPM 1 averaging intensity (time = 0)
0xE0000014	16	signed	BPM 2 averaging result (time = 0)
0xE0000016	16	unsigned	BPM 2 averaging variance * N * N_avg (time = 0)
0xE0000018	16	unsigned	BPM 2 averaging intensity (time = 0)
0xE000001A	16	signed	BPM 3 averaging result (time = 0)
0xE000001C	16	unsigned	BPM 3 averaging variance * N * N_avg (time = 0)
0xE000001E	16	unsigned	BPM 3 averaging intensity (time = 0)
0xE0000020	48	unsigned	time stamp, starting from gate high transition, 125 MHz (time = 1)
...	...	...	...

Table 6.5: BPM averaging result storage format

The BPM averaging result scope memory can hold up to 2^{24} samples. At a sampling frequency of 125 MHz, with a linear regression length of e.g. 1024 and with an averaging length of e.g. 1024 this corresponds to a maximum capture duration of 39.1 hours.

BPM {0 - 3} averaging result

This value divided by 2^{15} represents the relative beam position in the range $[-1, 1[$.

BPM {0 - 3} averaging variance * N * N_avg

This value divided by 2^{16} represents the variance of the corresponding BPM averaging result multiplied with the average linear regression length and the averaging length.

The scaling with the average linear regression length times the averaging length guarantees enough LSBs to evaluate. The variance itself quantized with 16 bits would otherwise nearly always result in zero.

BPM {0 - 3} averaging intensity

The average intensity of the beam. A value of $2^{16} - 1$ corresponds to the maximum achievable intensity at an alternating pattern of maximum and minimum ADC samples on both inputs.

6.3 Register map

6.3.1 Status registers

The following status registers can be read by software:

index	address	bits	radix	description
0	0x00000000	16	signed	latest BPM 0 result
1	0x00000008	16	signed	latest BPM 1 result
2	0x00000010	16	signed	latest BPM 2 result
3	0x00000018	16	signed	latest BPM 3 result
4	0x00000020	16	unsigned	latest BPM 0 variance * N
5	0x00000028	16	unsigned	latest BPM 1 variance * N
6	0x00000030	16	unsigned	latest BPM 2 variance * N
7	0x00000038	16	unsigned	latest BPM 3 variance * N
8	0x00000040	16	unsigned	latest BPM 0 intensity
9	0x00000048	16	unsigned	latest BPM 1 intensity
10	0x00000050	16	unsigned	latest BPM 2 intensity
11	0x00000058	16	unsigned	latest BPM 3 intensity
12	0x00000060	12	unsigned	effective linear regression length
13	0x00000068	48	unsigned	time from gate high transition
16	0x00000080	16	signed	latest BPM 0 averaging result
17	0x00000088	16	signed	latest BPM 1 averaging result
18	0x00000090	16	signed	latest BPM 2 averaging result
19	0x00000098	16	signed	latest BPM 3 averaging result
20	0x000000A0	16	unsigned	latest BPM 0 averaging variance * N * N_avg
21	0x000000A8	16	unsigned	latest BPM 1 averaging variance * N * N_avg
22	0x000000B0	16	unsigned	latest BPM 2 averaging variance * N * N_avg
23	0x000000B8	16	unsigned	latest BPM 3 averaging variance * N * N_avg
24	0x000000C0	16	unsigned	latest BPM 0 averaging intensity
25	0x000000C8	16	unsigned	latest BPM 1 averaging intensity
26	0x000000D0	16	unsigned	latest BPM 2 averaging intensity
27	0x000000D8	16	unsigned	latest BPM 3 averaging intensity
28	0x000000E0	12	unsigned	average effective linear regression length
32	0x00000100	2	unsigned	scope 0 capture status
33	0x00000108	32	unsigned	scope 0 next write address
40	0x00000140	2	unsigned	scope 1 capture status
41	0x00000148	32	unsigned	scope 1 next write address
48	0x00000180	2	unsigned	scope 2 capture status
49	0x00000188	32	unsigned	scope 2 next write address
124	0x000003E0	32	unsigned	build timestamp
125	0x000003E8	57	unsigned	FPGA serial number
126	0x000003F0	64	unsigned	module ID
127	0x000003F8	64	unsigned	magic number

Table 6.6: List of status registers

0 - 3: latest BPM {0 - 3} result

This value divided by 2^{15} represents the relative beam position in the range $[-1, 1[$.

4 - 7: latest BPM {0 - 3} variance * N

This value divided by 2^{16} represents the variance of the corresponding BPM result multiplied with the linear regression length.

The scaling with the linear regression length guarantees enough LSBs to evaluate. The variance itself quantized with 16 bits would otherwise often result in zero.

8 - 11: latest BPM {0 - 3} intensity

The intensity of the beam. A value of $2^{16} - 1$ corresponds to the maximum achievable intensity at an alternating pattern of maximum and minimum ADC samples on both inputs.

12: effective linear regression length

The effective linear regression length is determined by the nominal value of the linear regression length configuration register and the frequency of the RF pulses. If no RF pulses are present, this register should hold the value of the linear regression length configuration register.

13: time from gate high transition

Counter value, driven by the 125 MHz main processing clock, starting from the high transition of the gate input signal.

16 - 19: latest BPM {0 - 3} averaging result

This value divided by 2^{15} represents the relative beam position in the range $[-1, 1[$. Due to the averaging there should be less noise on this value than on the BPM result.

20 - 23: latest BPM {0 - 3} averaging variance * N * N_avg

This value divided by 2^{16} represents the variance of the corresponding BPM averaging result multiplied with the average linear regression length and the averaging length.

The scaling with the average linear regression length times the averaging length guarantees enough LSBs to evaluate. The variance itself quantized with 16 bits would otherwise nearly always result in zero.

24 - 27: latest BPM {0 - 3} averaging intensity

The intensity of the beam. A value of $2^{16} - 1$ corresponds to the maximum achievable intensity at an alternating pattern of maximum and minimum ADC samples on both inputs.

28: average effective linear regression length

The average of the effective linear regression lengths of the BPM.

32, 40, 48: Scope {0, 1, 2} capture status

value	capture status
0	idle
1	waiting for trigger
2	capturing
3	done

The value 0 is only present before starting the trigger for the first time. After that, the effective idle state is 3.

33, 41, 49: Scope {0, 1, 2} next write address

Address where the next data sample will be stored during the scope's capturing process.

124: build timestamp

Time when the bitstream was created. This information can be used to identify the gateware version (together with the Git commit information documented in https://git.gsi.de/BEA_HDL/FPGA_Common#122-gateware-information).

Format:

bits 31 - 27	bits 26 - 23	bits 22 - 17	bits 16 - 12	bits 11 - 6	bits 5 - 0
day	month	year (last two decimal digits)	hours	minutes	seconds

125: FPGA serial number

The XDMA PCIe driver by Xilinx numbers the devices randomly and is not able to identify the slot number of an AMC board. This register holds the FPGA's unique serial number and can be used to identify an AMC board.

126: module ID

The module ID can be used to identify the type of the current bitstream.
The module ID of the Cryring BPM gateware is `0x0102010300010001`.

The fields are defined as follows:

bits 63 - 56	bits 55 - 48	bits 47 - 40	bits 39 - 32	bits 31 - 16	bits 15 - 0
minor gateware version	major gateware version	minor board version	major board version	developer ID	project ID

Here is an incomplete list of project IDs:

project ID	project name
`0x0001`	Cryring BPM
`0x0002`	UniMon
`0x0003`	Rate Divider
`0x0004`	BLoFELD
`0x0005`	Resonant Transformer
`0x8001`	Red Pitaya

127: magic number

The magic number can be used to determine if the gateware uses the expected register format.
The value of this register is the same for all module IDs: `0xBADEAFFEDEADC0DE`.

6.3.2 Configuration registers

The following registers can be written by software:

index	address	bits	radix	description	default value
0	0x00000400	16	signed	ADC 0 offset correction summand	0x0000
1	0x00000408	16	signed	ADC 1 offset correction summand	0x0000
2	0x00000410	16	signed	ADC 2 offset correction summand	0x0000
3	0x00000418	16	signed	ADC 3 offset correction summand	0x0000
4	0x00000420	16	signed	ADC 4 offset correction summand	0x0000
5	0x00000428	16	signed	ADC 5 offset correction summand	0x0000
6	0x00000430	16	signed	ADC 6 offset correction summand	0x0000
7	0x00000438	16	signed	ADC 7 offset correction summand	0x0000
8	0x00000440	16	unsigned	ADC 0 gain correction factor	0x8000
9	0x00000448	16	unsigned	ADC 1 gain correction factor	0x8000
10	0x00000450	16	unsigned	ADC 2 gain correction factor	0x8000
11	0x00000458	16	unsigned	ADC 3 gain correction factor	0x8000
12	0x00000460	16	unsigned	ADC 4 gain correction factor	0x8000
13	0x00000468	16	unsigned	ADC 5 gain correction factor	0x8000
14	0x00000470	16	unsigned	ADC 6 gain correction factor	0x8000
15	0x00000478	16	unsigned	ADC 7 gain correction factor	0x8000
16	0x00000480	16	unsigned	BPM 0 capacitance correction factor	0x8000
17	0x00000488	16	unsigned	BPM 1 capacitance correction factor	0x8000
18	0x00000490	16	unsigned	BPM 2 capacitance correction factor	0x8000
19	0x00000498	16	unsigned	BPM 3 capacitance correction factor	0x8000
20	0x000004A0	12	unsigned	BPM linear regression length - 1	0x3FF
21	0x000004A8	5	unsigned	log2 of BPM averaging length	0x0A
22	0x000004B0	4	unsigned	gate signal input select	0x0
23	0x000004B8	4	unsigned	RF signal input select	0x8
24	0x000004C0	4	unsigned	intensity normalization exponent	0x0
25	0x000004C8	10	unsigned	moving average filter length - 1	0x000
26	0x000004D0	4	unsigned	IIR filter enable	0x0
32	0x00000500	26	unsigned	scope 0 capture length - 1	0x0000FFF
33	0x00000508	2	unsigned	scope 0 trigger mode	0x2
34	0x00000510	1	binary	scope 0 arm trigger	0
40	0x00000540	24	unsigned	scope 1 capture length - 1	0x000FFF
41	0x00000548	2	unsigned	scope 1 trigger mode	0x1
42	0x00000550	1	binary	scope 1 arm trigger	0
43	0x00000558	1	binary	scope 1 capture mode	0
48	0x00000580	24	unsigned	scope 2 capture length - 1	0x000FFF
49	0x00000588	2	unsigned	scope 2 trigger mode	0x1
50	0x00000590	1	binary	scope 2 arm trigger	0
51	0x00000598	1	binary	scope 2 capture mode	0
127	0x000007F8	1	binary	reset	0

Table 6.7: List of configuration registers

0 - 7: ADC {0 - 7} offset correction summand

Correction summand for a possible offset deviation of the ADC. The offset correction precedes the gain correction.

8 - 15: ADC {0 - 7} gain correction factor

Correction factor for a possible gain deviation of the ADC. The default value 0x8000 corresponds to a multiplication by 1. The possible correction range is $[0, 2[$.

16 - 19: BPM {0 - 3} capacitance correction factor

The capacitances of the two corresponding capacitor plates of a single BPM can differ. Data 0 is fed unchanged into the BPM algorithm, while data 1 is multiplied by a correction factor. The default value 0x8000 corresponds to a multiplication by 1. The possible correction range is $[0, 2[$.

20: BPM linear regression length - 1

Number of samples over which the linear regression is calculated if no external RF pulse signal is present. This value is valid for all four BPMs. If an external RF pulse signal is present, the result of the linear regression will be output and a new calculation will be started on every rising edge of the RF pulse signal. For this to work, this register has to be set to a value that is longer than the interval between the RF pulses.

Allowed values: 0x002 - 0xFFF

The lower limit is determined by the throughput of the divider IP core of 1 in 3 clock cycles that is used for the final division of the BPM algorithm.

21: Log2 of BPM averaging length

Dual logarithm of the number of linear regression results over which the averaging is calculated. This value is valid for all four BPMs.

Allowed range: 0 .. 20. Higher values will be set to the maximum allowed value. This corresponds to an averaging length of 1, 2, 4, ... , 1,048,576.

22: Gate signal input select

value	input
0 - 7	MLVDS line 0 - 7 on the backplane
8	FMC 0 *TRIG* input
9	FMC 1 *TRIG* input

The gate signal input can be switched between one of the eight MLVDS lines on the backplane and the two MMCX connectors labeled *TRIG* on the FMC front panels.

23: RF signal input select

value	input
0 - 7	MLVDS line 0 - 7 on the backplane
8	FMC 0 *TRIG* input
9	FMC 1 *TRIG* input

The RF signal input can be switched between one of the eight MLVDS lines on the backplane and the two MMCX connectors labeled *TRIG* on the FMC front panels.

24: Intensity normalization exponent

The intensity calculation in the BPM algorithm is normalized to ensure that no saturation can occur. This leads to small results during normal operation which are susceptible to quantization noise. By means of this register the normalization can be changed to allow larger results.

With each increment of this exponent by one, the result will double. Keep in mind that the result can saturate when setting this value to larger than zero.

25: Moving average filter length - 1

The averaging length minus 1 of the moving average filter on the ADC data. All possible values from 0 to 1023 are allowed, resulting in an averaging length between 1 and 1024.

25: IIR filter enable

Bitmask which enables the IIR filter per BPM. The filter is intended to suppress a 70 kHz interference on certain BPMs. Bit 0 enables the filter on the data of ADC 0 and 1, bit 1 on the data of ADC 2 and 3 and so on.

32, 40, 48: Scope {0, 1, 2} capture length - 1

The number of samples minus one that are stored after a scope has been triggered. Each sample consists of 16 bytes.

Scope 0 can only handle even numbers of samples. Uneven numbers will be automatically handled as the next higher even number. For scopes 1 and 2, also uneven numbers are allowed.

33, 41, 49: Scope {0, 1, 2} trigger mode

value	trigger mode
0	trigger on rising edge of gate signal
1	trigger on high state of gate signal
2, 3	trigger instantly after the trigger is armed, independent of the state of the gate signal

34, 42, 50: Scope {0, 1, 2} arm trigger

Writing a 1 to this register will arm the trigger once. The register does not have to be reset to 0 before the next arm trigger, just write another 1 to it. If the corresponding register *continuous trigger* is set to 1, writing to this register does not have any effect.

When in 'waiting for trigger' state (see status register 'capture status' in chapter 6.2.1), writing a 0 to this register will cancel the arming of the trigger and the capture status will change to 'done'.

43, 51: Scope {1, 2} capture mode

value	capture mode
0	capture until the number of samples defined by register {40, 48} are stored
1	the same, but cancel capturing when the gate signal goes low

A capture mode register is only available for scopes 1 and 2. Scope 0 (for corrected ADC data) always operates in capture mode 0.

127: Reset

Writing a 1 to this register triggers a reset on the gateware, which also resets all configuration registers to their default values.
The reset will be automatically lifted so that the register does not have to be written to 0 after initiating a reset.

6.4 Capturing procedure

6.4.1 Known number of samples

A typical procedure for capturing a predefinable number of samples starting from the rising edge of the gate signal is the following:

- write the number of samples minus 1 to the configuration register *capture length - 1*
- write a 0 to the configuration register *trigger mode*
- write a 0 (= default) to the configuration register *capture mode*
- write a 1 to the configuration register *arm trigger*
- you can check the status register 'capture status' for the progress: `1`: rising edge of gate signal not yet detected, `2`: capturing is ongoing, `3`: capturing completed
- you can check the current write address by polling the status register *next write address*

6.4.2 Unknown number of samples

BPM results are only calculated while the gate signal is high. If you want to capture a complete high period of e.g. BPM average samples, the total number of samples is unknown. Proceed as follows:

- write the maximum value 0x1FFFFFF to the configuration register *capture length - 1*
- write a 0 to the configuration register *trigger mode*
- write a 1 to the configuration register *capture mode*
- write a 1 to the configuration register *arm trigger*
- you can check the status register *capture status* as above
- the value of the status register *next write address* will be static after completion and indicates how many samples have been captured

7 Extended gateware software interface

Besides the interface documented in chapter 6 which is meant for productive use, there is an extended interface for development and debugging purposes. The extended interface is also present in the bitstream by default.

While the productive interface is intended to be kept as downward compatible as possible, the extended interface may be subject to major changes during the development process.

7.1 Extended register map

7.1.1 Additional status registers

The following additional status registers can be read by software:

index	address	bits	radix	description
64	0x00000200	1	binary	FMC 0 SPI busy
65	0x00000208	8	unsigned	FMC 0 SPI read data
66	0x00000210	1	binary	FMC 0 I2C busy
67	0x00000218	8	unsigned	FMC 0 I2C read data
68	0x00000220	1	binary	FMC 0 PLL status
69	0x00000228	38	unsigned	FMC 0 VCXO initial RFREQ
70	0x00000230	38	unsigned	FMC 0 VCXO RFREQ
71	0x00000238	32	unsigned	FMC 0 measured ADC clock frequency
72	0x00000240	32	unsigned	FMC 0 ADC FIFO underflow counter
73	0x00000248	32	unsigned	FMC 0 ADC FIFO overflow counter
80	0x00000280	1	binary	FMC 1 SPI busy
81	0x00000288	8	unsigned	FMC 1 SPI read data
82	0x00000290	1	binary	FMC 1 I2C busy
83	0x00000298	8	unsigned	FMC 1 I2C read data
84	0x000002A0	1	binary	FMC 1 PLL status
85	0x000002A8	38	unsigned	FMC 1 VCXO initial RFREQ
86	0x000002B0	38	unsigned	FMC 1 VCXO RFREQ
87	0x000002B8	32	unsigned	FMC 1 measured ADC clock frequency
88	0x000002C0	32	unsigned	FMC 1 ADC FIFO underflow counter
89	0x000002C8	32	unsigned	FMC 1 ADC FIFO overflow counter
96	0x00000300	16	unsigned	ADC 0 max peak to peak
97	0x00000308	16	unsigned	ADC 1 max peak to peak
98	0x00000310	16	unsigned	ADC 2 max peak to peak
99	0x00000318	16	unsigned	ADC 3 max peak to peak
100	0x00000320	16	unsigned	ADC 4 max peak to peak
101	0x00000328	16	unsigned	ADC 5 max peak to peak
102	0x00000330	16	unsigned	ADC 6 max peak to peak
103	0x00000338	16	unsigned	ADC 7 max peak to peak
111	0x00000378	1	binary	SDRAM initial calibration complete

Table 7.1: List of additional status registers

64, 80: FMC {0, 1} SPI busy

Indicates that a SPI read or write access is going on. The value of this register has to be checked to be 0 before triggering a SPI access.

65, 81: FMC {0, 1} SPI read data

Contains the result of a read access to a SPI register.

66, 82: FMC {0, 1} I2C busy

Indicates that an I2C read or write access is going on. The value of this register has to be checked to be 0 before triggering an I2C access.

67, 83: FMC {0, 1} I2C read data

Contains the result of a read access to an I2C register.

68, 84: FMC {0, 1} PLL status

Value of the configurable output pin *status* of the AD9510 PLL and clock distribution IC. By default this pin indicates lock status of the PLL.

69, 85: FMC {0, 1} VCXO initial RFREQ

RFREQ is a factory calibrated multiplicator to the XTAL frequency of the Si571 programmable VCXO. Before the programming of a new output frequency this value has to be read (see chapter 4.1).

69, 86: FMC {0, 1} VCXO RFREQ

The VCXO output frequency is programmed to 125 MHz by the gateware. This register holds the value of *RFREQ* that has been programmed (see chapter 4.1).

71, 87: FMC {0, 1} measured ADC clock frequency

The ADC clock is measured against the main processing clock. This register holds the number of detected ADC clock cycles during 1 second of the main processing clock.

72, 88: FMC {0, 1} ADC FIFO underflow counter

If the ADC clock is slower than the main processing clock, samples will be repeated by the clock domain crossing FIFO output logic. For each repetition the underflow counter will be incremented by 1.

73, 89: FMC {0, 1} ADC FIFO underflow counter

If the ADC clock is faster than the main processing clock, samples will discarded by the clock domain crossing FIFO input logic. For each discarded sample the overflow counter will be incremented by 1.

{96 - 103}: ADC {0 - 7} max peak to peak

The maximum and the minimum value of the ADC data is determined over a free running period of 1 second. This register contains the difference of the maximum and the minimum value.

111: SDRAM initial calibration complete

The communication to the SDRAM is controlled by an IP core by Xilinx which performs a timing calibration at start up. The value of this register will be 1 after completion of the initial calibration.

7.1.2 Additional configuration registers

The following additional registers can be written by software:

index	address	bits	radix	description	default value
27	0x000004D8	1	binary	ADC test data enable	0
39	0x00000538	1	binary	scope 0 continuous trigger	0
47	0x00000578	1	binary	scope 1 continuous trigger	0
55	0x000005B8	1	binary	scope 2 continuous trigger	0
56	0x000005C0	1	binary	AFC LED select	0
57	0x000005C8	3	unsigned	AFC LED value	0x7
58	0x000005D0	1	binary	gate override	0
59	0x000005D8	1	binary	gate override value	1
60	0x000005E0	8	unsigned	MLVDS direction	0x00
61	0x000005E8	8	unsigned	MLVDS output value	0x00
64	0x00000600	2	unsigned	FMC 0 status LED select	0x0
65	0x00000608	3	unsigned	FMC 0 status LED value	0x7
66	0x00000610	5	unsigned	FMC 0 SPI chip select	0x0F
67	0x00000618	1	binary	FMC 0 SPI read/write	0
68	0x00000620	8	unsigned	FMC 0 SPI address	0x00
69	0x00000628	8	unsigned	FMC 0 SPI write data	0x00
70	0x00000630	1	binary	FMC 0 SPI trigger	0
71	0x00000638	1	binary	FMC 0 ADC resetn	1
72	0x00000630	1	binary	FMC 0 I2C read/write	0
73	0x00000638	7	unsigned	FMC 0 I2C device address	0x49
74	0x00000650	8	unsigned	FMC 0 I2C register address	0x00
75	0x00000658	8	unsigned	FMC 0 I2C write data	0x00
76	0x00000660	1	binary	FMC 0 I2C trigger	0
77	0x00000668	1	binary	FMC 0 PLL resetn	1
78	0x00000670	1	binary	FMC 0 clock switch select	1
79	0x00000678	1	binary	FMC 0 VCXO output enable	1
80	0x00000680	2	unsigned	FMC 1 status LED select	0x0
81	0x00000688	3	unsigned	FMC 1 status LED value	0x7
82	0x00000690	5	unsigned	FMC 1 SPI chip selec	0x0F
83	0x00000698	1	binary	FMC 1 SPI read/write	0
84	0x000006A0	8	unsigned	FMC 1 SPI address	0x00
85	0x000006A8	8	unsigned	FMC 1 SPI write data	0x00
86	0x000006B0	1	binary	FMC 1 SPI trigger	0
87	0x000006B8	1	binary	FMC 1 ADC rstn	1
88	0x000006C0	1	binary	FMC 1 I2C read/write	0
89	0x000006C8	7	unsigned	FMC 1 I2C device address	0x49
90	0x000006D0	8	unsigned	FMC 1 I2C register address	0x00
91	0x000006D8	8	unsigned	FMC 1 I2C write data	0x00
92	0x000006E0	1	binary	FMC 1 I2C trigger	0
93	0x000006E8	1	binary	FMC 1 PLL rstn	1
94	0x000006F0	1	binary	FMC 1 clock switch select	1
95	0x000006F8	1	binary	FMC 1 VCXO output enable	1

Table 7.2: List of additional configuration registers - part 1

index	address	bits	radix	description	default value
96	0x00000700	5	unsigned	FMC 0 ADC 0 clock delay	0x0D
97	0x00000708	5	unsigned	FMC 0 ADC 1 clock delay	0x0D
98	0x00000710	5	unsigned	FMC 0 ADC 2 clock delay	0x0D
99	0x00000718	5	unsigned	FMC 0 ADC 3 clock delay	0x0D
100	0x00000720	5	unsigned	FMC 0 ADC 0 data delay	0x00
101	0x00000728	5	unsigned	FMC 0 ADC 1 data delay	0x00
102	0x00000730	5	unsigned	FMC 0 ADC 2 data delay	0x00
103	0x00000738	5	unsigned	FMC 0 ADC 3 data delay	0x00
104	0x00000730	5	unsigned	FMC 1 ADC 0 clock delay	0x0D
105	0x00000738	5	unsigned	FMC 1 ADC 1 clock delay	0x0D
106	0x00000750	5	unsigned	FMC 1 ADC 2 clock delay	0x0D
107	0x00000758	5	unsigned	FMC 1 ADC 3 clock delay	0x0D
108	0x00000760	5	unsigned	FMC 1 ADC 0 data delay	0x00
109	0x00000768	5	unsigned	FMC 1 ADC 1 data delay	0x00
110	0x00000770	5	unsigned	FMC 1 ADC 2 data delay	0x00
111	0x00000778	5	unsigned	FMC 1 ADC 3 data delay	0x00

Table 7.3: List of additional configuration registers - part 2

17: ADC test data enable

If set to 1, all eight ADC data inputs will be overridden by a counter value which is incremented by 1 every 125 MHz clock cycle.

39, 47, 55: Scope {0, 1, 2} continuous trigger

If set to 1, the trigger is armed and will be rearmed automatically after every capture completion.

56: AFC LED select

There is one tricolor LED at the center of the AFC front panel labeled *L3* that can be controlled by the gateware.

value	input
0	PCIe reference clock, blink frequency divided by 2^{27}, white
1	static value from register 113 'AFC LED value'

57: AFC LED value

Static lighting pattern if register 112 'AFC LED select' = 1.

bit	color
0	red
1	green
2	blue

58: Gate override

For testing purposes without an external gate signal this register can be set to 1 to simulate a gate signal via the register 115 'gate override value'.

59: Gate override value

Can be used to simulate a gate signal when register 114 'gate override' is 1.

60: MLVDS direction

Determines the direction of the eight MLVDS lines on the AMC connector. A '0' corresponds to an input to the FPGA and a '1' to an output from the FPGA.

61: MLVDS output value

Determines the logic levels of the eight MLVDS lines if they are configured as outputs (see previous register).

64, 80: FMC {0, 1} status LED select

There is one tricolor LED on the FMC front panel labeled *status* that can be controlled by the gateware.

value	input
0	ADC clock, blink frequency divided by 2^{27}, green if AD9510 PLL is in lock, otherwise red
1	AD9510 monitoring clock, blink frequency divided by 2^{27}
2, 3	static value from register 'status LED value'

65, 81: FMC {0, 1} status LED value

The static lighting pattern defined by this register becomes active if the corresponding register 'status LED select' is set to 2 or 3.

bit	color
0	red
1	green
2	blue

66, 82: FMC {0, 1} SPI cs

Chip select signals (active high) of the SPI bus to the four ADCs and to the AD9510 PLL and clock distribution.

bit	device
0	ADC 0
1	ADC 1
2	ADC 2
3	ADC 3
4	PLL and clock distribution

67, 83: FMC {0, 1} SPI read/write

0: write mode, 1: read mode

68, 84: FMC {0, 1} SPI address

The address of the register that shall be accessed.

69, 85: FMC {0, 1} SPI write data

The data that shall be written to a register.

70, 86: FMC {0, 1} SPI trigger

Write a 1 to this register to start a read or write access on the SPI bus. The register does not have to be reset to 0 before the next SPI trigger, just write another 1 to it.

71, 87: FMC {0, 1} ADC resetn

Low active reset signal to the four ADCs in parallel. Tie to 0 and back to 1 to initiate a reset.

72, 88: FMC {0, 1} I2c read/write

0: write mode, 1: read mode

73, 89: FMC {0, 1} I2C device address

The address of the connected VCXO is 0x49.

74, 90: FMC {0, 1} I2C register address

The address of the register that shall be accessed.

75, 91: FMC {0, 1} I2C write data

The data that shall be written to a register.

76, 92: FMC {0, 1} I2C trigger

Write a 1 to this register to start a read or write access on the I2C bus. The register does not have to be reset to 0 before the next I2C trigger, just write another 1 to it.

77, 93: FMC {0, 1} PLL resetn

Low active reset signal to the PLL and clock distribution. Tie to 0 and back to 1 to initiate a reset.

78, 94: FMC {0, 1} Clock switch select

There is a separate clock switch in front of the AD9510 PLL reference clock input.

value	connect to
0	MMCX connector labeled *REF* on the front panel of the FMC board
1	clock output from the FPGA via the FMC connector

79, 95: FMC {0, 1} VCXO output enable

Enables the frequency output of the VCXO.

96 - 99 and 104 - 107: FMC {0, 1} ADC {0 - 3} clock delay

There is a configurable input delay for setting the correct digital interface timing for both the clock and the data signals. Increasing this value increases the delay of the clock, so that the data is sampled later.

100 - 103 and 108 - 111: FMC {0, 1} ADC {0 - 3} data delay

See above. Increasing this value increases the delay of the data, so that the data is sampled at an earlier position.

8 Hardware properties

8.1 LEDs driven by the FPGA gateware

There are three tricolor LEDs connected to FPGA pins:

- *L3* in the center of the AFC front panel: Currently displays the PCIe reference clock divided by 2^{27} in white.
- *LD1* (v1.0) or *STATUS* (v1.2 and v2.3) on the right of the two FMC board's front panels: Currently display the ADC clock frequencies divided by 2^{27} in green if the PLLs indicate a lock, otherwise red.

Each tricolor LED consists of three independent LEDs (red, green and blue).

8.2 Differences between FMC ADC 250 M 16B 4CH versions

The *LD1* (v1.0) or *STATUS* (v1.2 and v2.3) LED on the right of the FMC board front panel is connected differently between v1.0 and (v1.2 and v2.3). When using the location constraints for v1.2 and v2.3 together with a v1.0 board, the LED lights as follows:

- wanted red: off
- wanted green: lights green
- wanted blue: lights red

Also, the MMCX input *TRIG* seems to be connected differently on v1.0. Feeding HF-Pulses into v1.0 boards does not work with the current bitstream.

8.3 Analog characteristics

8.3.1 ADC input filter

The FMC ADC boards were originally designed for very high input frequencies and are equipped with input filters that show a pronounced high pass characteristic. There are different versions of the boards which have a different ADC input filter circuitry.

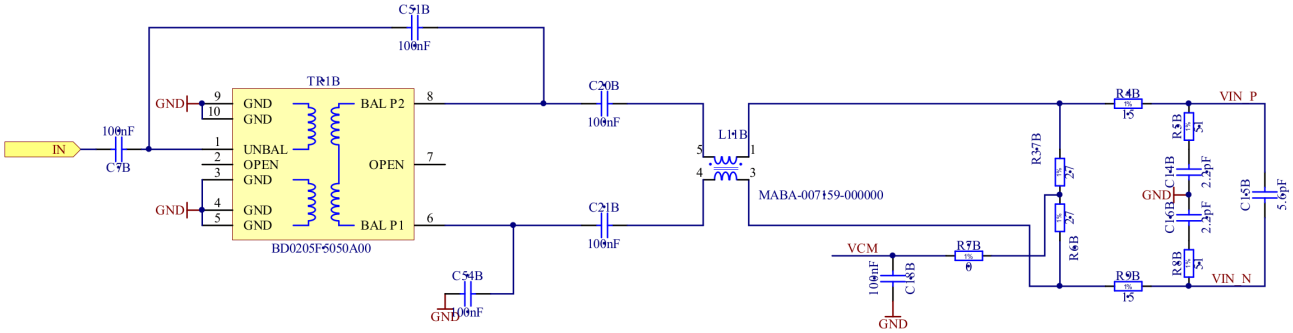

Figure 8.1: Schematics of the original ADC input filter of versions 1.0 and 1.2. Image taken from [12]

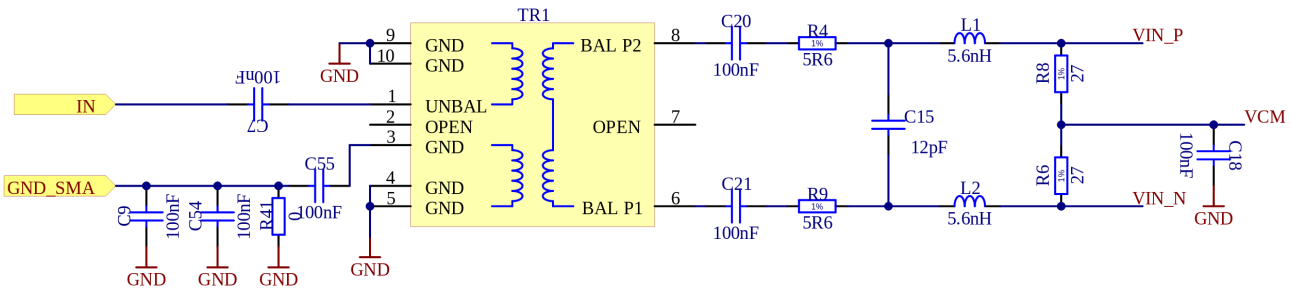

Figure 8.2: Schematics of the original ADC input filter of version 2.3. Image taken from [13]

The part labeled *TR1(B) BD0205F5050A00* is a balun with an operating range of 70 - 1000 MHz [14]. Lower frequencies are severely attenuated.

For being able to use the FMC ADC boards in the Cryring BPM system, the baluns have to be replaced by more suitable components.

Two approaches have been implemented on versions 1.0 and 1.2 (probably by Piotr Miedzik):

1. each balun is replaced by two wires
2. each balun is replaced by two capacitors of probably 100 nF (hint in an old email)

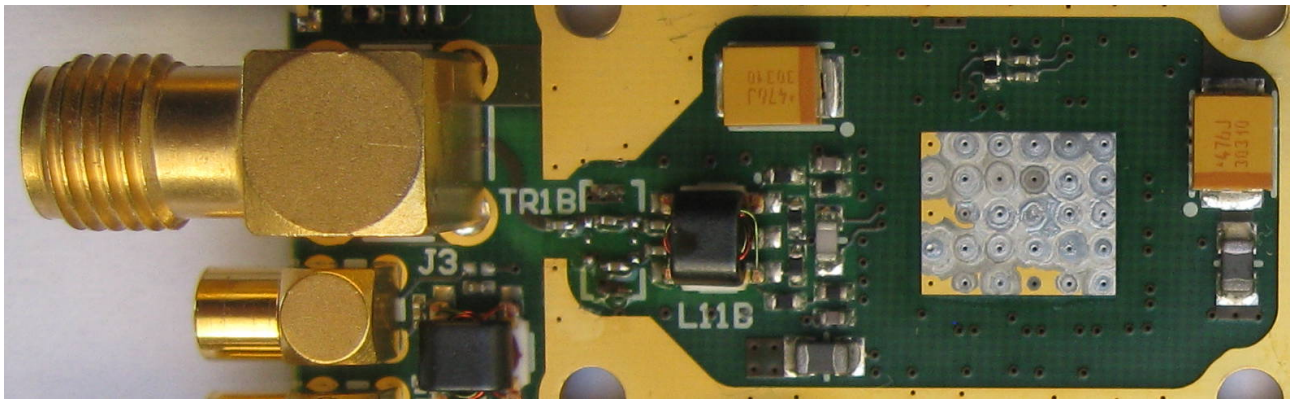

Figure 8.3: v1.2 ADC input filter: balun TR1B replaced by two capacitors

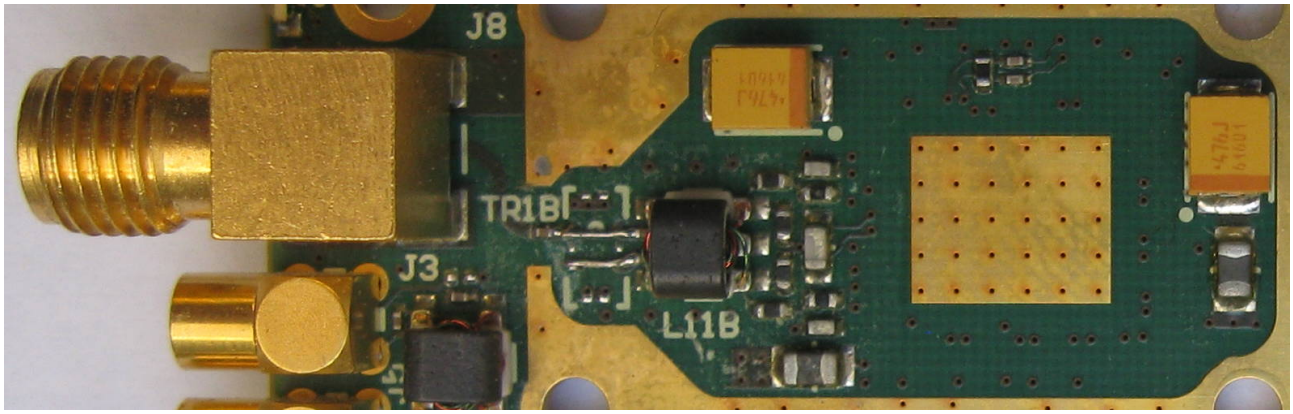

Figure 8.4: v1.0 ADC input filter: balun TR1B replaced by two wires

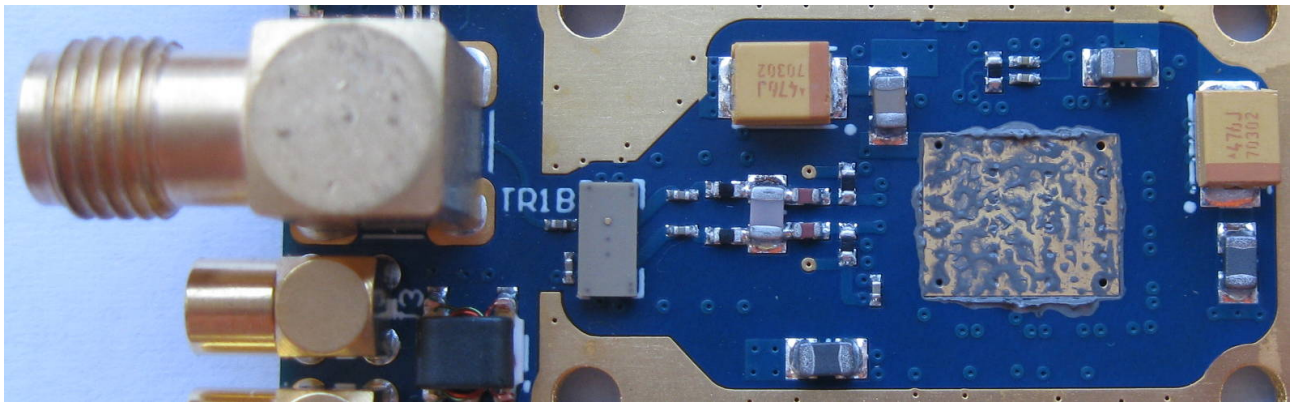

Figure 8.5: Original v2.3 ADC input filter

The heatspreader under the bottom of the FMC ADC board has to be unscrewed to access the baluns.

There is a significant difference in the ADC input filter circuitry between versions 1.0 and 1.2 and version 2.3. In version 2.3 the transmission line transformers *L11 {A, B, C, D}* have been removed and the RC filter has been modified.

Figure 8.6 shows the magnitude frequency responses of the original v2.3 input filter and of the two modifications of the v1.0 and v1.2 input filters. The diagram data was created by using a sine signal from a signal generator with an amplitude of 2 V_{pp} and by measuring the maximum amplitude swing of the raw ADC data.

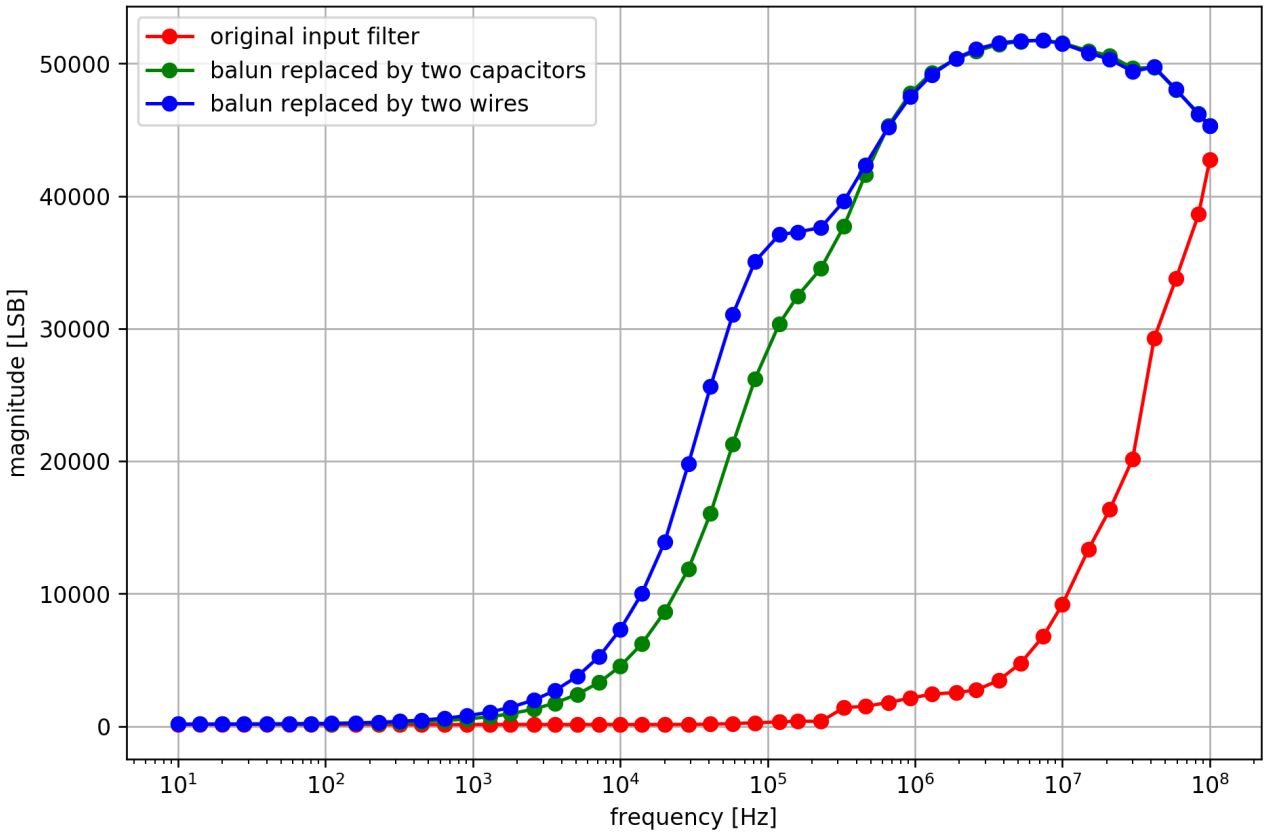

Figure 8.6: Magnitude frequency responses of different ADC input filters

8.4 Required changes for PLL lock

For being able to drive the reference clock of the PLLs on the FMC ADC boards from an FPGA output pin, a pullup resistor indicating the clock direction has to be desoldered. The resistor is labeled *R132* in the FMC ADC schematic [13]. Figure 8.7 shows the location of the resistor.

Without this change slight clock frequency differences between the processing clock on the AFC board and the ADC clocks on the FMC boards occur. There are synchronization FIFOs which ensure correct clock domain crossings, but in this case a small fraction of ADC samples might have to be discarded or repeated once, depending on which frequency is higher. This will always affect all the samples of a FMC board in parallel, so that no differences between the two input data streams of a single BPM will occur and no measurable effect on the BPM results should be observed.

With this changes applied and with the correct settings of the PLL, the ADC samples of both FMC boards will arrive exactly in parallel and no samples will be lost or repeated.

Update: Even though removing the resistor is advisable, the PLL also locks with it in place. Not all the boards in the productive setup have been changed.

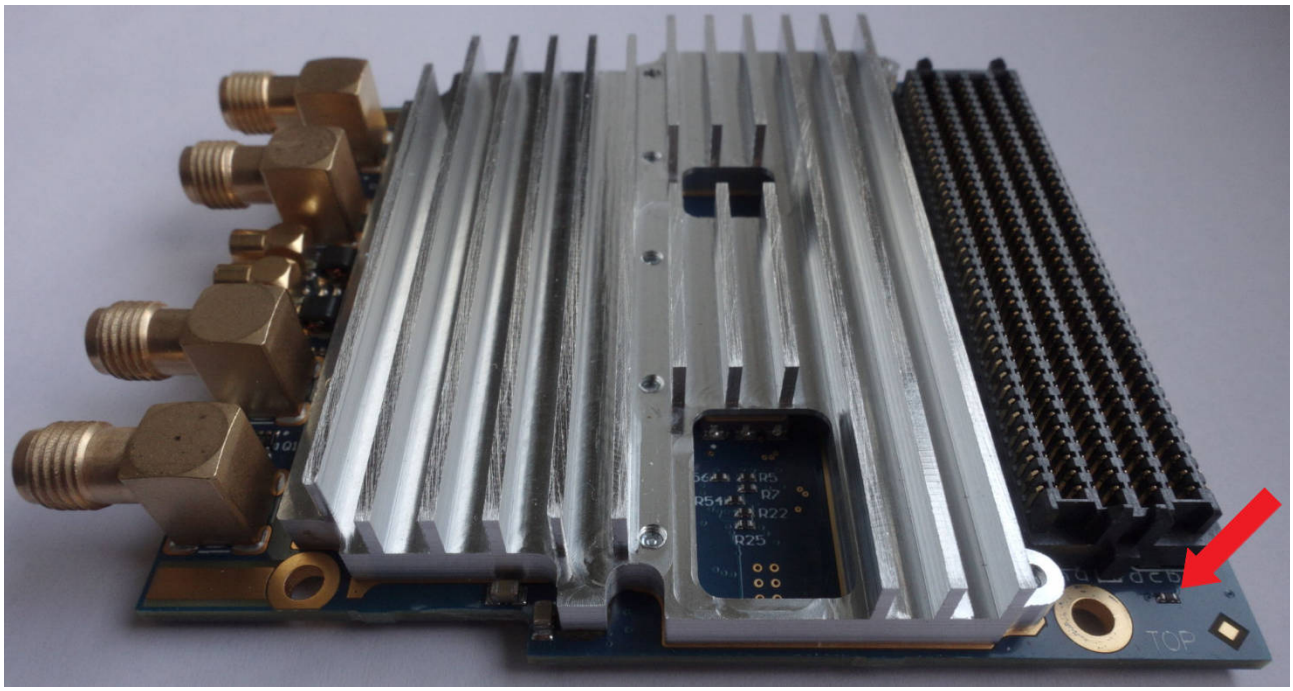

Figure 8.7: Pullup resistor which needs to be removed for the PLL to lock

8.5 List of ADC-FMC boards

Label	AFC number	FMC number	remarks
ADC 1	4	2	from old setup
ADC 2			from old setup, port 3 has lower amplitude (0.8) and a phase shift
ADC 3	1	2	from old setup
ADC 4			from old setup, works fine
ADC 5	4	1	from old setup
ADC 6			from old setup, port 3 has an offset of about -2000 LSBs
ADC 7			from old setup, port 2 has higher amplitude (1.15) and a phase shift
ADC 8			from old setup, port 3 does not work, port 2 equal to that of ADC 7
ADC 9			from old setup, not tested
ADC 10	1	1	old spare
ADC 11	5	2	old spare
ADC 12	2	1	new
ADC 13	2	2	new
ADC 14	3	1	new
ADC 15	5	1	new

8.6 Productive setup

The following AFC version 3.1 boards are installed in the productive setup in the Cryring container:

AFC number	FPGA serial number	FMC 1	FMC 2	AMC slot
1	0x048A82110D1B05C	ADC 10	ADC 3	3
2	0x008182110D1B05C	ADC 12	ADC 13	4
3	0x010A82110D1B05C	ADC 14	-	5
4	0x068B48160E47054	ADC 5	ADC 1	6
5	0x018D5C24235885C	ADC 15	ADC 11	7

The FPGA serial number is not printed anywhere, but can only be read from the DNA_PORT primitive by the gateware. In this gateware the FPGA serial number is read out and stored in a register (see chapter 6.2.1).

The signal cables are connected as follows:

signal	AFC number	FMC number	port number
YR2DX1HL	1	1	1
YR2DX1HR	1	1	2
YR2DX2VO	1	1	3
YR2DX2VU	1	1	4
YR10DX1HL	1	2	1
YR10DX1HR	1	2	2
YR10DX2VO	1	2	3
YR10DX2VU	1	2	4
YR3DX3VO	2	1	3
YR3DX3VU	2	1	4
YR3DX4HL	2	1	1
YR3DX4HR	2	1	2
YR12DX1HL	2	2	1
YR12DX1HR	2	2	2
YR12DX2VO	2	2	3
YR12DX2VU	2	2	4
YR7DX1HL	3	1	1
YR7DX1HR	3	1	2
YR7DX2VO	3	1	3
YR7DX2VU	3	1	4
YR3DX1HL	4	1	1
YR3DX1HR	4	1	2
YR3DX2VO	4	1	3
YR3DX2VU	4	1	4
YR8DX1HL	4	2	1
YR8DX1HR	4	2	2
YR8DX2VO	4	2	3
YR8DX2VU	4	2	4
YR6DX1HL	5	1	1
YR6DX1HR	5	1	2
YR6DX2VO	5	1	3
YR6DX2VU	5	1	4
YR11DX1HL	5	2	1
YR11DX1HR	5	2	2
YR11DX2VO	5	2	3
YR11DX2VU	5	2	4

9 Test coverage

9.1 BPM algorithm

9.1.1 Simulation

This test simulates the VHDL code of the gateware and is automatically run by the CI/CD pipelines of Gitlab.

All ADC data inputs are driven by the same repeated pattern of positive and negative values, but with different amplitudes.

For the same patterns on both inputs of a BPM with the amplitudes A_0 and A_1,

δ_i can be expressed as $c \cdot \sigma_i$ with $c = \frac{A_0 - A_1}{A_0 + A_1}$

and equation 4.1 simplifies to:

$$x = \frac{N \sum_i \sigma_i \delta_i - (\sum_i \sigma_i)(\sum_i \delta_i)}{N \sum_i \sigma_i^2 - (\sum_i \sigma_i)^2} = c = \frac{A_0 - A_1}{A_0 + A_1} \tag{9.1}$$

so that the expected BPM result can be calculated as:

$x = 2^{15} \cdot \frac{A_0 - A_1}{A_0 + A_1}$

BPM	ADC	relative amplitudes A_0, A_1	expected BPM result	simulated BPM result
0	0	1	10922.67	10922
	1	1/2		
1	2	1/2	-10922.67	-10923
	3	1		
2	4	1	0	0
	5	1		
3	6	1	25486.22	25487
	7	1/8		

The simulation results are consistent with the expectations considering possible numeric calculation deviations in the numerous calculation steps, which might influence the least significant bits.

9.1.2 Using a function generator as data source

Digital gain setting

The following measurement was made using a function generator which was configured to output two phase aligned sines with an amplitude of $2V_{pp}$ which were connected to the two inputs of a BPM.

The linear regression length and the averaging length were both set to 1024.

Before starting the measurement, the digital gain of one of the two ADC inputs was corrected so that the BPM averaging result equaled 0.

After that, the digital gain correction of the other ADC input was used to set different amplitudes in order to avoid possible nonlinearities of the function generator gain.

The results were read from the FPGA Observer GUI which displays the BPM results divided by 2^{15}.

relative amplitude	expected BPM result	measured BPM result
1/8	$0.\overline{7}$	0.778
2/8	0.6	0.600
3/8	$0.\overline{45}$	0.454
4/8	$0.\overline{3}$	0.333
5/8	0.2308	0.230
6/8	0.1429	0.142
7/8	$0.0\overline{6}$	0.066
8/8	0	0.000

The measurement results are consistent with the expectations considering noise and possible numeric calculation deviations in the numerous calculation steps, which might influence the least significant bits.

9.2 Reliability tests

A test of the BPM scope and the BPM averaging scope was run overnight. The two inputs of a BPM were fed by two function generator outputs with the following settings:

- sine signal
- frequency: 1 MHz
- output to ADC0: 2.001 V_{pp}
- output to ADC1: 0.667 V_{pp}

According to equation **??**, the expected BPM result for the chosen input amplitudes is 0.5.

The test was run for 12 hours, during which the scopes were read continuously and histograms of the occurring results were created.

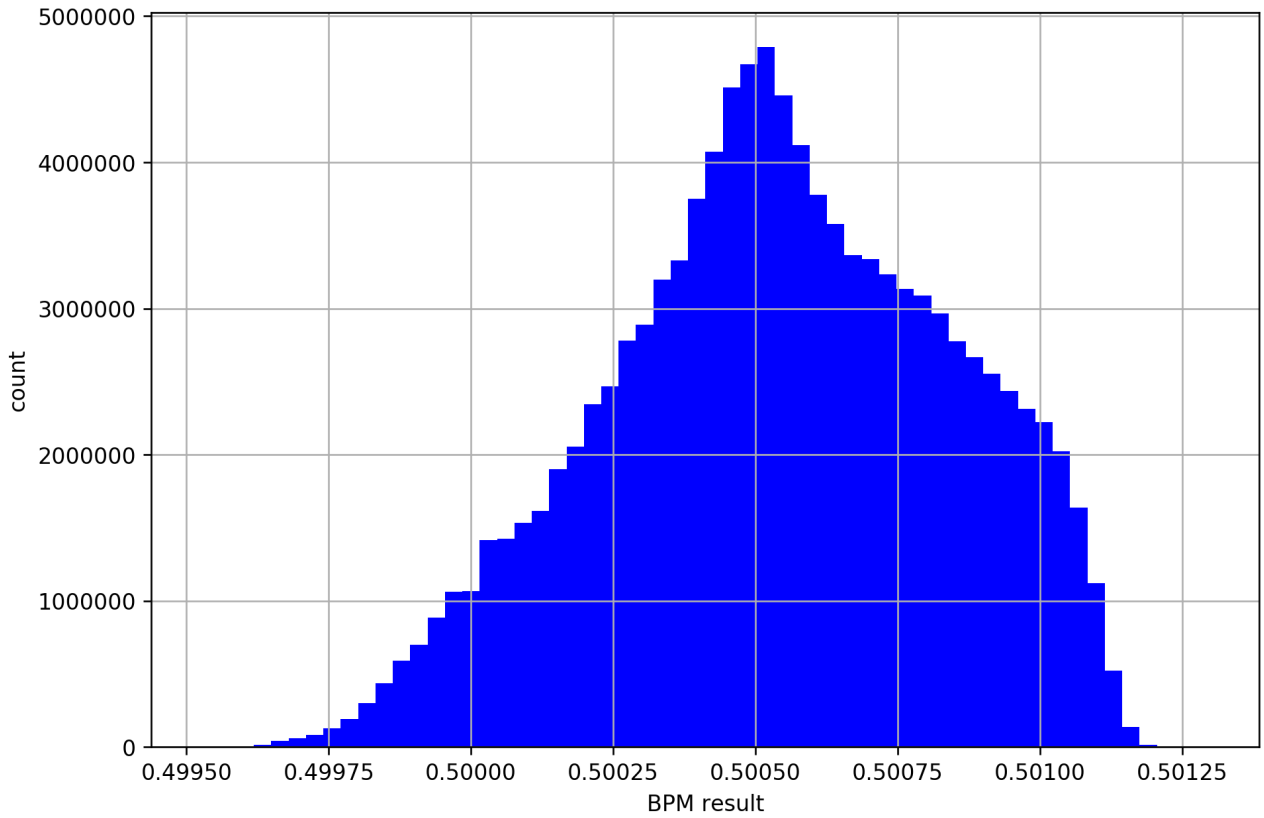

Figure 9.1: BPM result histogram

The gateware was configured as follows:

- ADC1 gain correction: 0x8398 BPM averaging result = 0.5
- linear regression length: 1024
- averaging length: 1024
- number of samples per capture: 1024

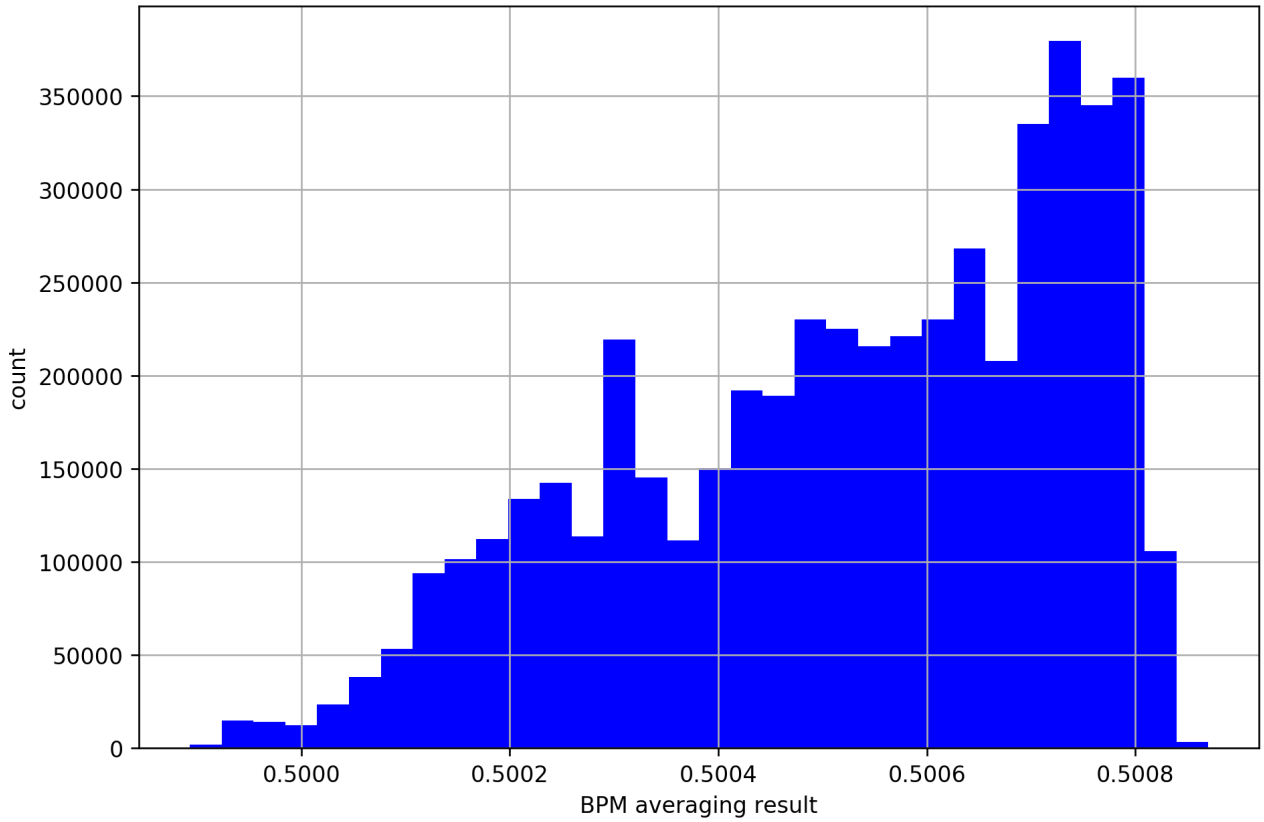

Figure 9.2: BPM averaging result histogram

The resulting histograms do not show Gaussian distributions, but still seem to be reasonably confined. The deviations from Gaussian distributions might have been caused by temperature shifts during the night which might have affected the amplitudes.

References

[1] A. Reiter, R. Singh: Comparison of beam position calculation methods for application in digital acquisition systems. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, February 2018, https://git.gsi.de/BEA_HDL/datasheets/-/blob/master/Paper_BPM_Algorithm.pdf

[2] P. Miedzik, H. Bräuning, T. Hoffmann, A. Reiter, R. Singh: A MicroTCA based beam position monitoring system at CRYRING@ESR. *16th Int. Conf. on Accelerator and Large Experimental Control Systems, Barcelona, Spain*, October 2017, https://git.gsi.de/BEA_HDL/datasheets/-/blob/master/Paper_BPM_architecture.pdf

[3] A. Reiter, W. Kaufmann, R. Singh, P. Miedzik, T. Hoffmann, H. Bräuning: The CRYRING BPM cookbook, March 2015, https://git.gsi.de/BEA_HDL/datasheets/-/blob/master/Cryring_BPM_System_Overview.pdf

[4] AMC FMC Carrier (AFC) Git repository, Open Hardware Repository, https://ohwr.org/project/afc/wikis/home

[5] Xilinx: 7 Series FPGAs Data Sheet: Overview, https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[6] Silicon Labs: Timing part decoder web page, https://www.silabs.com/timing/lookup-customize

[7] Silicon Labs: Si571 datasheet, https://www.silabs.com/documents/public/data-sheets/si570.pdf

[8] Analog Devices: AD9510 datasheet, https://www.analog.com/media/en/technical-documentation/data-sheets/AD9510.pdf

[9] Renesas: ISLA216P datasheet, https://www.renesas.com/us/en/www/doc/datasheet/isla216p.pdf

[10] Xilinx: Artix 7 datasheet: DC and AC Switching Characteristics, https://www.xilinx.com/support/documentation/data_sheets/ds181_Artix_7_Data_Sheet.pdf

[11] AFC v3.1 schematics, https://git.gsi.de/BEA_HDL/datasheets/-/blob/master/Schematics_AFC_v3.1.pdf

[12] FMC ADC 250M 16B 4ch v1.2 schematics, https://git.gsi.de/BEA_HDL/datasheets/-/blob/master/Schematics_FMC_ADC_250M_16B_4ch_v1_2.pdf

[13] FMC ADC 250M 16B 4ch v2.3 schematics, https://github.com/lnls-dig/fmc250-hw/blob/master/circuit_board/ADC.SchDoc

[14] Anaren: Balun BD0205F5050A00 datasheet, https://git.gsi.de/BEA_HDL/datasheets/-/blob/master/Balun_BD0205F5050A00_datasheet.pdf