European XFEL Cavity BPM bpm_cav_exfel Data Sheet

Content

Table of Contents

Figures

1 Introduction

For the E-XFEL a cavity BPM was designed consisting of the 3.3 GHz cavity pickup, RFFE board, ADC16HL sampling board, the GPAC carrier board and additional boards (transition cards, EVR) supporting the application. The aim is to measure sub um resolution of the beam position.
The firmware described here is used to sample the I/Q amplitudes of x and y (beam position dependent signals) and reference channel (beam charge dependent signal) of the Cavity RFFE and calculate the position with minimal latency in a Virtex-5 FXT FPGA. Low latency is important for feedback systems relying on beam position information.

1.1 Purpose

The aim of this document is to provide a global overview of the measurement technique with the GPAC and describe the user interface of the BPM FPGAs on the GPAC.

1.2 Scope

This document provides a detailed overview of the firmware interface and specifies the user interface.

1.3 Definitions, acronyms, and abbreviations

This document is based on the “IEEE Recommended Practice for Software Requirements Specifications” [1].

ADC	Analog Digital Converter.
BPM	Beam Position Monitor
bunch train	All bunches belonging to one macro-pulse (macro-pulse as defined in the specification of DESY timing system)
EVR	EVent Receiver. Decoder of the Event Link at PSI. This is a standard PSI VME card used to decode event link messages for trigger generation.
FPGA	Field Programmable Gate Array. Programmable logic device.
reg	Register. Mathematically z^-1
RTM	Rear transition card. Sometimes called “Transition Card” or “Transition Module”
TBD	To Be Defined. Means has to be defined by the user at setup of the FEL.

1.4 References

[1] IEEE Std 830-1998, Recommended Practice for Software Requirements Specifications.
[2] PSI 2012, Cavity BPM ADC Data Processing Ideas, Cavity_BPM_ADC_Data_Processing_Ideas_v1r4.pdf
..\..\..\..\..\..\04_Data_Sheet\01_Reference\Cavity_BPM_ADC_Data_Processing_Ideas_v1r4.pdf

[3] Xilinx December 2 2008, Virtex-5 FPGA Data Sheet: DC and Switching Characteristics the Virtex5 datasheet DS202

1.5 Overview

Chapter 2 provides an overview and how the firmware is related to other firmware used. Chapter 3 contains all the detail information on the user interfaces.

2 Overall description

Figure 1 illustrates the main devices used for the BPM system. On the left hand side the pickups are drawn, which are connected to the RFFE. Within the RFFEs the 3.3 GHz signals from the pickups are mixed with a machine RF derived frequency to obtain a pulse long enough to be measured with the six 16 bit 160 MSa/s ADC on the ADC16HL piggyback board. The ADC data is processed in the BPM FPGA on the GPAC and is finally provided to the control system.

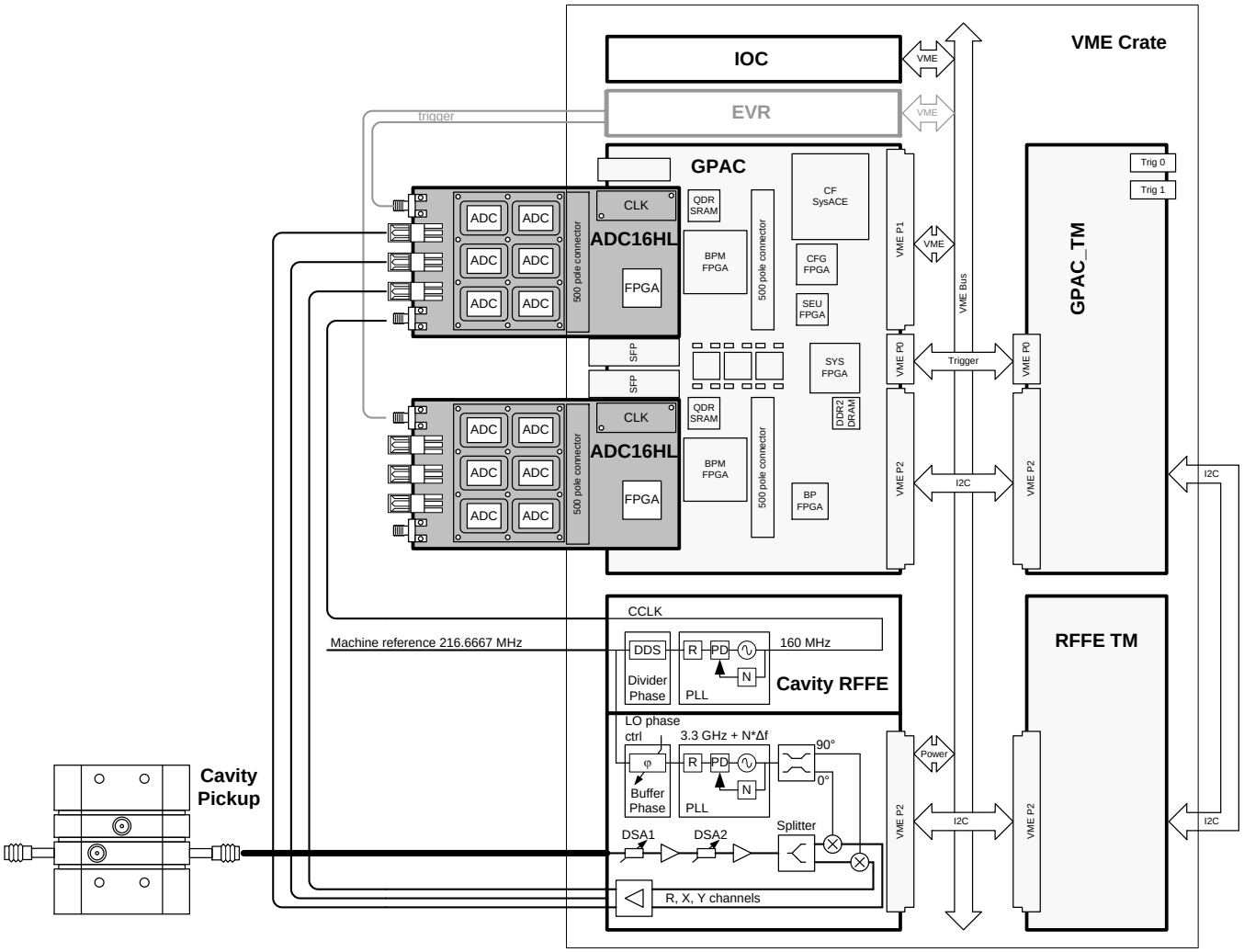

Figure 1: Cavity BPM Overview

The control system configures the BPM system by means of providing signals and information concerning the timing of the beam (a very stable machine RF and event system containing bunch train trigger, bunch number, etc.), the charge and filling pattern expected. This information has to be sent to the BPM system in advance, e.g. before the bunch train is fired.

3 Machine Interface Overview

The control (event) system of the E-XFEL machine will provide a signal denoted in this document as a bunch train trigger. This bunch train trigger informs the BPM that after a location dependant delay (B1.0) the bunch train will start; containing a predefined number of bunches (B0.0).

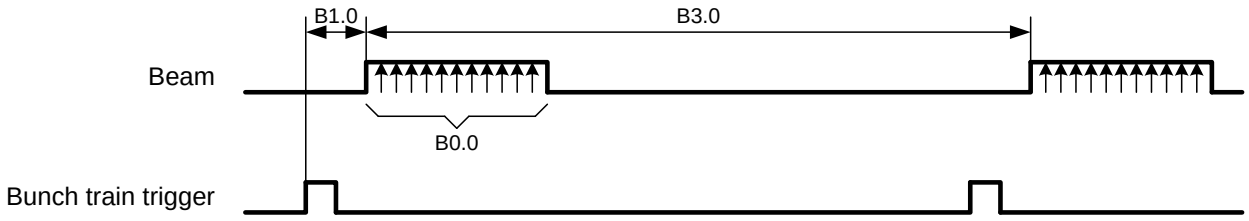

Figure 2: Machine Interface for the BPM System

Time	Value	Description
B0.0	1…3072	Number of bunches in the bunch train. The BPM electronics will measure and calculate this amount of positions.
B1.0	Location dependent	Delay between event transmitted on the event link cable (timing system of the machine) and the beam measured by a particular pickup. Stable delay, but depends on the position in the machine, hence different for each BPM
B3.0	max. 30 Hz	Bunch train repetition rate. This is a machine parameter, e.g. can be regarded as minimal delay between bunch trains.

The bunch train trigger is encoded in an event link of the E-XFEL.

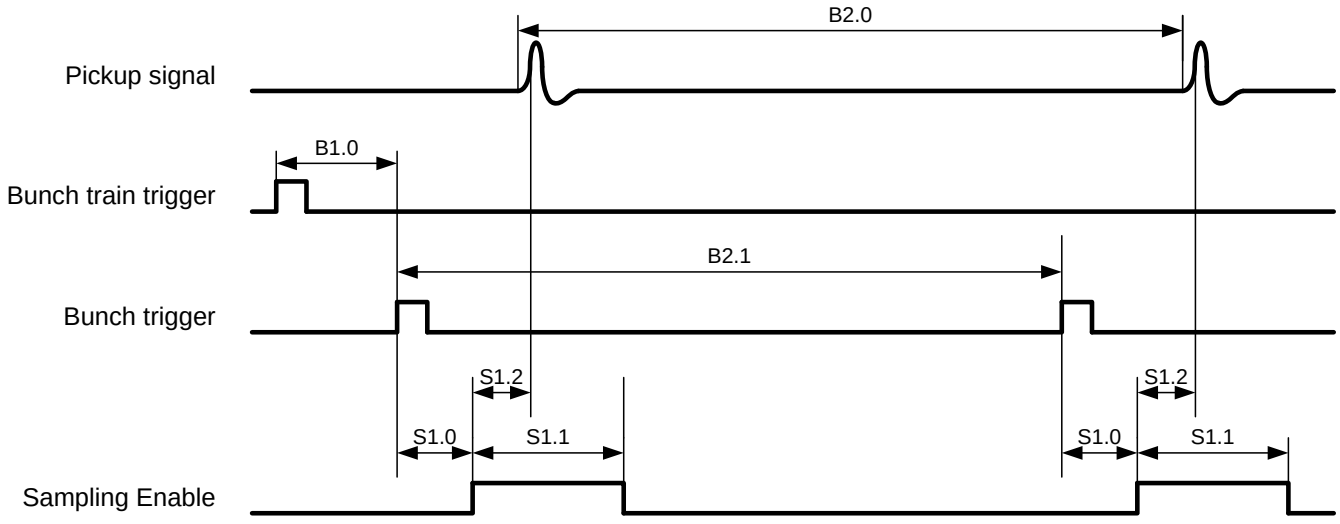

Figure 3: Machine Interface for the BPM FPGA

Time	Value	Description
B1.0	Location dependent	Delay between event transmitted on the event link cable (timing system of the machine) and the beam measured by a particular pickup. Stable delay, but depends on the position in the machine, hence different for each BPM.
B2.0	200 ns … 1 ms	Bunch repetition rate. This is a machine setting carried out by the control

		system.
B2.1	200 ns … 1 ms in FPGA clock cycles	Basically the same as the bunch repetition rate but implemented as a counter in the BPM FPGA, hence the units are clock cycles. This is a machine setting carried out by the control system.
S1.0	TBD by the user	Sampling delay for the beam measurement.
S1.1	TBD by the user	Sampling duration for the beam measurement. In order to reduce the latency the user has to try to keep this window as short as possible.
S1.2	TBD by the user	Pre-samples of the beam measurement. This value is important if the bunch train trigger is missing and the FPGA needs to find the beam within the ADC data stream. The setting is only relevant in the signal (self) triggered mode.

3.1 Firmware Interface Overview

The processing of the digitized data is carried out in the BPM FPGA of the GPAC in a firmware component called “bpm_cav_exfel”.
The processing firmware consists of several major blocks that are summarized with more detail on the following pages for the implementation of a control system interface:

The data adjustment (adc_smp_data_adjust) block is needed in order to fulfill the setup and hold time requirements of the FPGA input flip-flops due to the feature of setting individual sampling clock delays for each ADC channel. For example, the individual sampling clock phase matches with the data rise and fall times and hence the FPGA needs to move the data sampling window away from these data changes into a stable region in order to consistently retrieve the data.
A finite state machine (adc_smp_fsm) manages this core. The state machine receives the mode of operation from the user application and controls the various blocks in order to provide the desired functionality (external/self triggered, delays, number of samples, etc.). The timing requirements are also implemented in this block. For example: In order to move the trigger in time a programmable delay (B1.0) is implemented. This decouples the external trigger requirement to match exactly the occurrence of the trigger to the signal being measured.

The individual ADCs will provide measurement data with an average that is non zero. This difference is denoted as “baseline” in this document. The base line (adc_smp_baseline) can be subtracted automatically from the measurement values in order to adjust for reference level fluctuations due to various effects (disturbance of other channels, pile up effects, temperature, offset of the RFFE amplifier, ADC variations, etc.). The user can select if the calculated baseline of each channel shall be subtracted or not. The calculated baseline is provided to the user as raw data (base line calc), which is a feature that indicates the noise level on the cables.

The calculation block (adc_smp_calc) carries out the main processing. For the BPM project a sophisticated calculation engine is required that resides in this block [3]. In this block the position calculation is performed in addition to the correction of I/Q imbalance, pickup rotation and physical unit scaling.

Finally the consistently sampled data is stored in memory (adc_smp_buffer and adc_smp_bram) and can be read by the user. The outputs of the storages are connected to software accessible memory address spaces that in turn are readable by the control system as single values or bursts transfers (DMA).

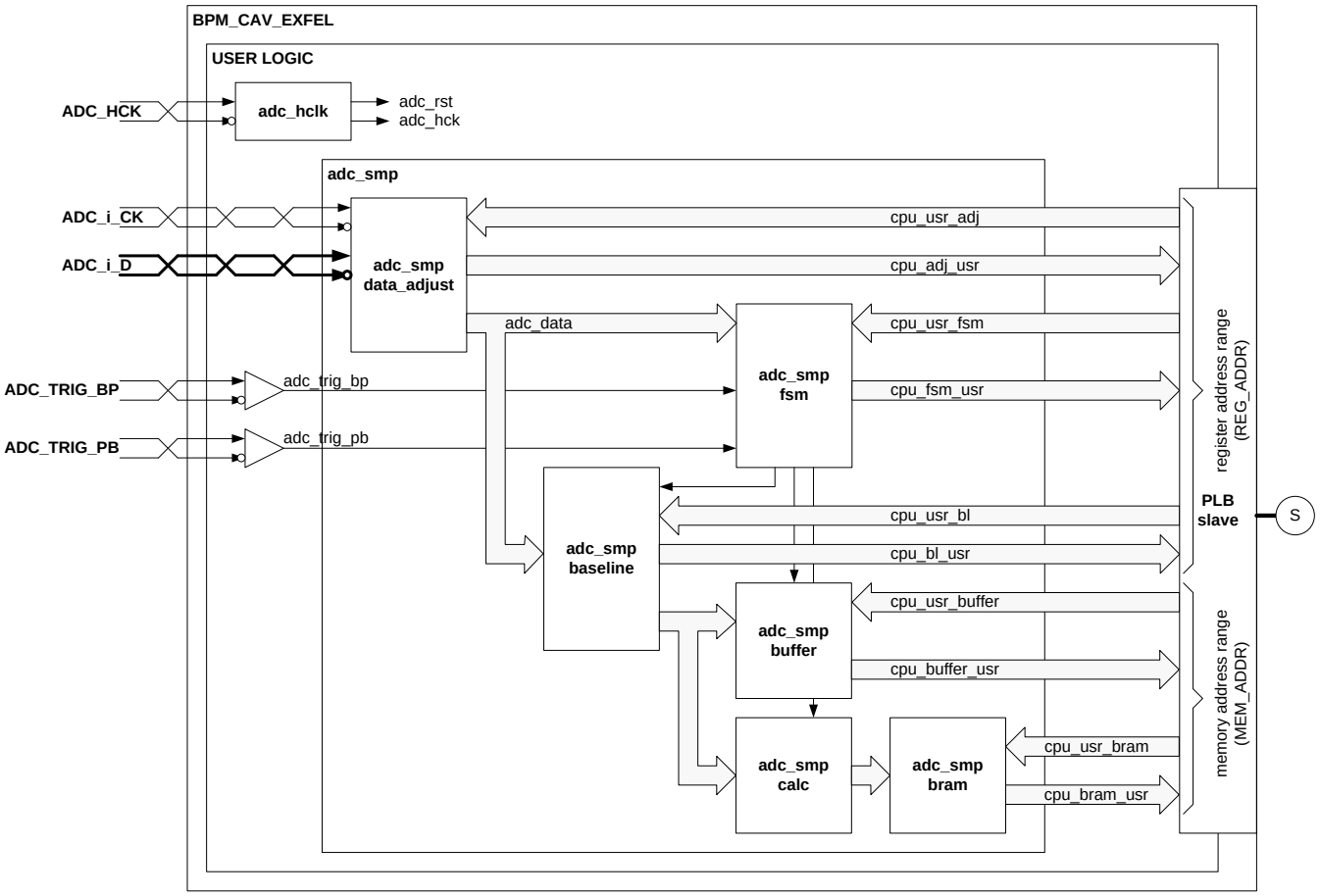

Figure 4: Block Diagram of bpm_cav_exfel

Please note: The block and signal names in the figure above are used as component names and signal structures in the vhdl source code. The character “i” in some of the signal names has to be substituted with the numbers (ADC channels) 0 to 5 in order to be found in the vhdl source code. If the trace denotes a differential signal, then a “_p” and “_n” have to be appended to the appropriate signal name.
It is important to remember that there is a register address range and a memory address range. Make sure you understand the difference after reading this document.

4 BPM_CAV_EXFEL Firmware User Logic interface

The following chapters will refer to the figure “Block Diagram of bpm_cav_exfel” and define further the functionality down to a register description allowing a controls system programmer to make use of the functionality provided by hardware and firmware.

Important Note: The following description of the registers and bits used in the register is based on a 32 bit wide word being accessed by means of an address. The address is a byte-address hence the address increments by 4 between two adjacent registers, with the MSB being accessed by byte address + 0 and the LSB being accessed by address + 3! (Big Endian Machine). Please ensure this is correct when programming the control interface with data types others than 32 bits.
In addition the register base address and the memory base address have to be distinguished because they denote two different positions in the memory address map (Xilinx EDK tool feature). Be careful that you add the register or memory offset to the addresses correctly.

4.1 ADC_HCLK

The FPGA on the GPAC obtains the processing clock from the ADC16HL card. This clock (adc_hck) is used to run one of the two clock domains in the bpm_cav_exfel firmware. If during the configuration of the ADC16HL clock distribution invalid clocks are generated, then the programmable logic in turn generates a reset (adc_rst). This allows the firmware to operate again in a defined/intended frequency range avoiding odd FPGA behavior.

4.2 ADC_SMP_DATA_ADJUST

The ADC data arrives at the BPM FPGA on the GPAC with a different phase compared to the local clock adc_hck, due to routing delays and the programmable sampling clock delay in the clock distribution chip of the ADC16HL. In order to sample the data consistently, timing requirement for setup (tIDOCK) and hold (tIOCKD) time of the Virtex5 [3] input flip-flops, the ADC inputs are shifted by delay elements in the FPGA to a valid pattern. The GPAC PCBs were designed such that the routing delays, including the on chip flight times (FPGA) are equalized. Hence the ADC channels form groups of traces of equalized length. Therefore the relevant setup and hold times for signals arriving at the GPAC are the ones defined by the Virtex5 datasheet [3] in chapter “Input/Output Logic Switching Characteristics Table 60: ILOGIC Switching Characteristics”. This input delay is controlled by the adc_smp_data_adjust block.

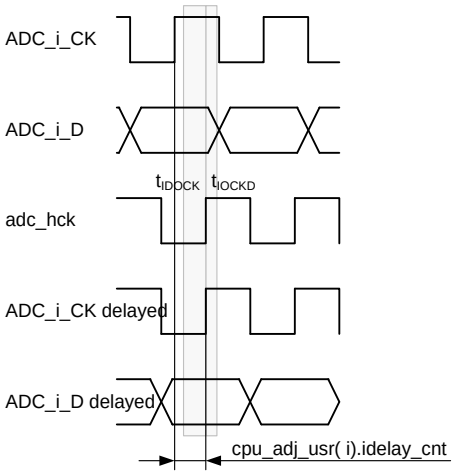

Figure 5: ADC_SMP_DATA_ADJUST Timing Diagram

The Virtex5 FPGA input delay (IDELAY primitive) has a resolution of 78 ps per step measured by cpu_adj_usr(i).idelay_cnt. Hence the data lines and adjacent clocks of each ADC are delayed in these steps.

Address (REG_ADDR)	R/W	Bit	Name	Description
0x00000040	W	any	cpu_usr_adj(0). start	Start adjusting the delay for ADC(0). 'X' = Any write starts the adjusting of the delay
0x00000040	R	0	cpu_adj_usr(0). done	Delay busy indicator ADC(0). '0' = Busy, delay adjustment is in progress '1' = Adjustment completed
		21:16	cpu_adj_usr(0). idelay_cnt	Delay set in steps of 78 ps for ADC(0)
0x00000044	W	any	cpu_usr_adj(1). start	Start adjusting the delay for ADC(1). 'X' = Any write starts the adjusting of the delay
0x00000044	R	0	cpu_adj_usr(1). done	Delay busy indicator ADC(1). '0' = Busy, delay adjustment is in progress '1' = Adjustment completed
		21:16	cpu_adj_usr(1). idelay_cnt	Delay set in steps of 78 ps for ADC(1)
0x00000048	W	any	cpu_usr_adj(2). start	Start adjusting the delay for ADC(2). 'X' = Any write starts the adjusting of the delay
0x00000048	R	0	cpu_adj_usr(2). done	Delay busy indicator ADC(2). '0' = Busy, delay adjustment is in progress '1' = Adjustment completed
		21:16	cpu_adj_usr(2). idelay_cnt	Delay set in steps of 78 ps for ADC(2)
0x0000004C	W	any	cpu_usr_adj(3). start	Start adjusting the delay for ADC(3). 'X' = Any write starts the adjusting of the delay
0x0000004C	R	0	cpu_adj_usr(3). done	Delay busy indicator ADC(3). '0' = Busy, delay adjustment is in progress '1' = Adjustment completed
		21:16	cpu_adj_usr(3). idelay_cnt	Delay set in steps of 78 ps for ADC(3)
0x00000050	W	any	cpu_usr_adj(4). start	Start adjusting the delay for ADC(4). 'X' = Any write starts the adjusting of the delay
0x00000050	R	0	cpu_adj_usr(4). done	Delay busy indicator ADC(4). '0' = Busy, delay adjustment is in progress '1' = Adjustment completed

		21:16	cpu_adj_usr(4). idelay_cnt	Delay set in steps of 78 ps for ADC(4)
0x00000054	W	any	cpu_usr_adj(5). start	Start adjusting the delay for ADC(5). 'X' = Any write starts the adjusting of the delay
0x00000054	R	0	cpu_adj_usr(5). done	Delay busy indicator ADC(5). '0' = Busy, delay adjustment is in progress '1' = Adjustment completed
		21:16	cpu_adj_usr(5). idelay_cnt	Delay set in steps of 78 ps for ADC(5)

Table 1: ADC_SMP_DATA_ADJUST Register Map

4.3 ADC_SMP_FSM

The adc_smp_fsm is the main control for all blocks in the bpm_cav_exfel firmware. The most important signal is the cpu_usr_fsm.ena which enables the whole state machine for retrieving data. The cpu_usr_fsm.mode defines whether the sampling of the ADC values shall be based on a particular ADC value (cpu_usr_fsm.trig_val) or after receiving an external trigger input ADC_TRIG_BP/ADC_TRIG_PB after a delay (cpu_usr_fsm.B1_0 + cpu_usr_fsm.S1_0). The number of samples is programmable (cpu_usr_fsm.S1_1) and defines the waveform length stored in the buffer. After all samples are stored in the buffer an indication is raised to start reading the data (cpu_fsm_usr.read_ready).
A useful feature is the delay (cpu_fsm_usr.trig_dly) in units of clock cycles between the external trigger (ADC_TRIG_BP/ADC_TRIG_PB) and the threshold value of the ADC data (cpu_usr_fsm.trig_val). In a stable FEL timing and event system the external trigger is expected to be synchronized with the machine and hence with the ADC data being sampled. This is especially useful in BPM systems due to the reference cavity providing a position independent and comparatively large signal.

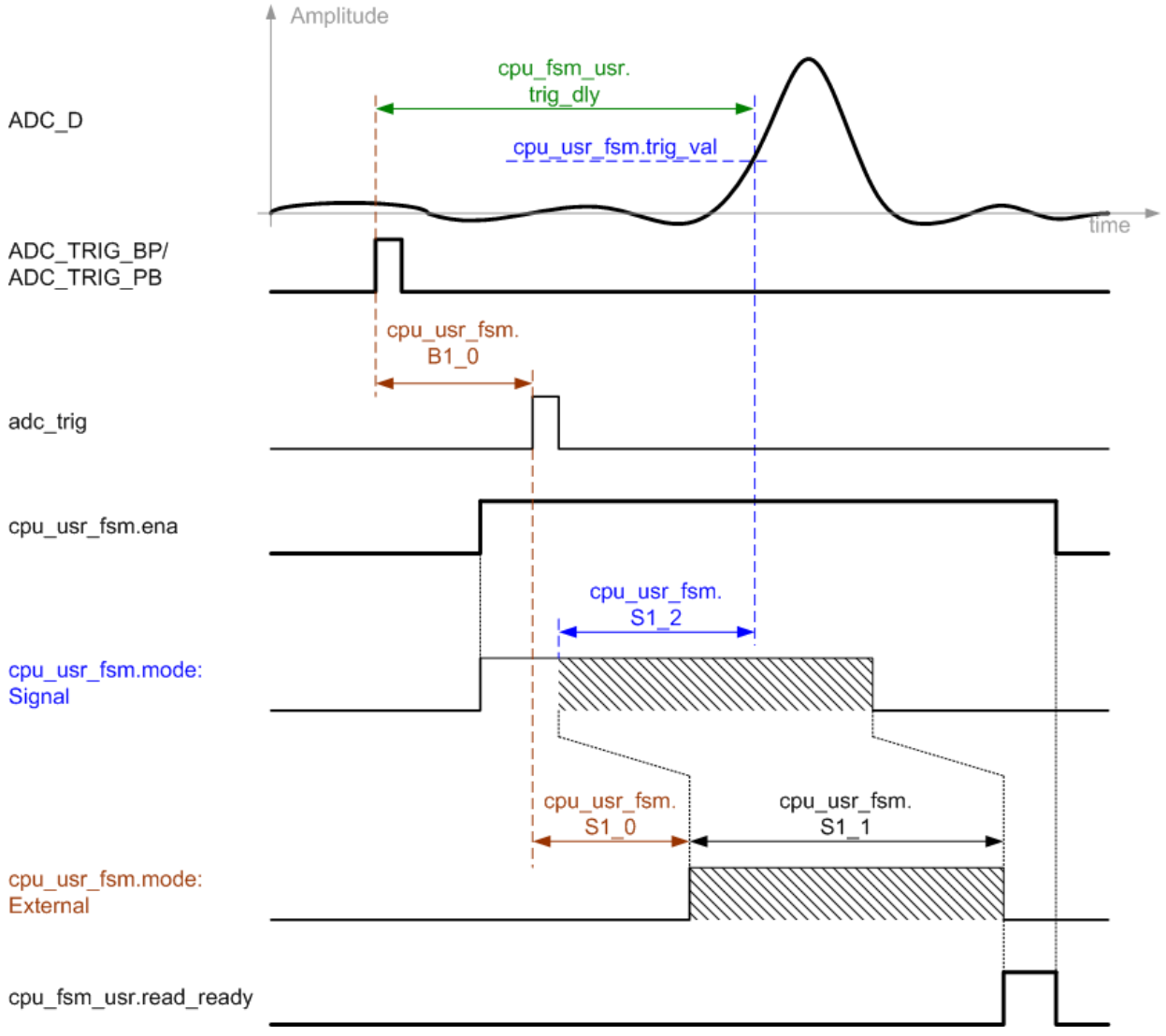

Figure 6: ADC_SMP_FSM Timing Diagram

Address (REG_ADDR)	R/W	Bit	Name	Description
0x00000000	R/W	0	cpu_usr_fsm. ena	FSM enable. '0' = Put FSM to idle state. '1' = Start process of sampling data of a bunch train after triggering (external or signal trigger).
		8	cpu_usr_fsm. mode	FSM mode 0 = External Trigger (which means external trigger connected to the GPAC BPM FPGA, this should be the bunch train trigger) 1 = Signal Trigger (Which means to trigger on a reference channel value threshold)
0x00000004	R	0	cpu_fsm_usr. read_ready	FSM status ‘0’ = FSM is currently busy or disabled ‘1’ = FSM has finished sampling the data and completed the calculation
0x00000010	R/W	15:0	cpu_usr_fsm. B0_0	Number of bunches expected. Corresponds to B0.0
0x00000014	R/W	31:0	cpu_usr_fsm. B1_0	Number of clock cycles delay if mode is in “External Trigger”. Corresponds to B1.0
0x00000018	R/W	15:0	cpu_usr_fsm. B2_0	Clock cycles between bunches. Corresponds to B2.0
0x0000001C	R/W	31:0	cpu_usr_fsm. B3_0	Timeout for external and signal trigger. After this time in units of clock cycles the trigger is issued and starts reading the samples. Corresponds to B3.0
0x00000020	R/W	15:0	cpu_usr_fsm. S1_0	Number of clock cycles delay between the internal bunch trigger (adc_trig) until start of processing the sampled data.
0x00000024	R/W	15:0	cpu_usr_fsm. S1_1	Number of ADC samples requested per bunch.
0x00000028	R/W	4:0	cpu_usr_fsm. S1_2	Presamples. If cpu_usr_fsm.mode is set to mode “Signal Triggered” then this setting defines the number of samples priors the threshold. The sampling is triggered when a rising edge of the signal is detected (similar to an oscilloscope) with a predefined number of presamples. This setting here is basically the number of presamples required.
0x0000002C	R/W	15:0	cpu_usr_fsm. trig_val	Threshold. If mode (cpu_usr_fsm.mode) is in “Signal Trigger”. The sampling is processed when a rising edge of the signal is detected (similar to an oscilloscope) crossing this threshold.
0x00000038	R	31:0	cpu_fsm_usr. trig_cnt	Count the trigger occurrences in the configured mode (cpu_usr_fsm.mode).
0x0000003C	R	31:0	cpu_fsm_usr. trig_dly	Number of clock cycles between ADC_TRIG_xy and ADC value has a rising edge bigger than cpu_usr_fsm.trig_val

Table 2: ADC_SMP_FSM Register Map

4.4 ADC_SMP_BASELINE

The sampled ADC data is used to calculate a sliding average over a programmable number of samples (2^cpu_usr_bl.samples). This average is denoted as baseline. The average can be delayed (cpu_usr_bl.delay) before subtracted from the current ADC data values.

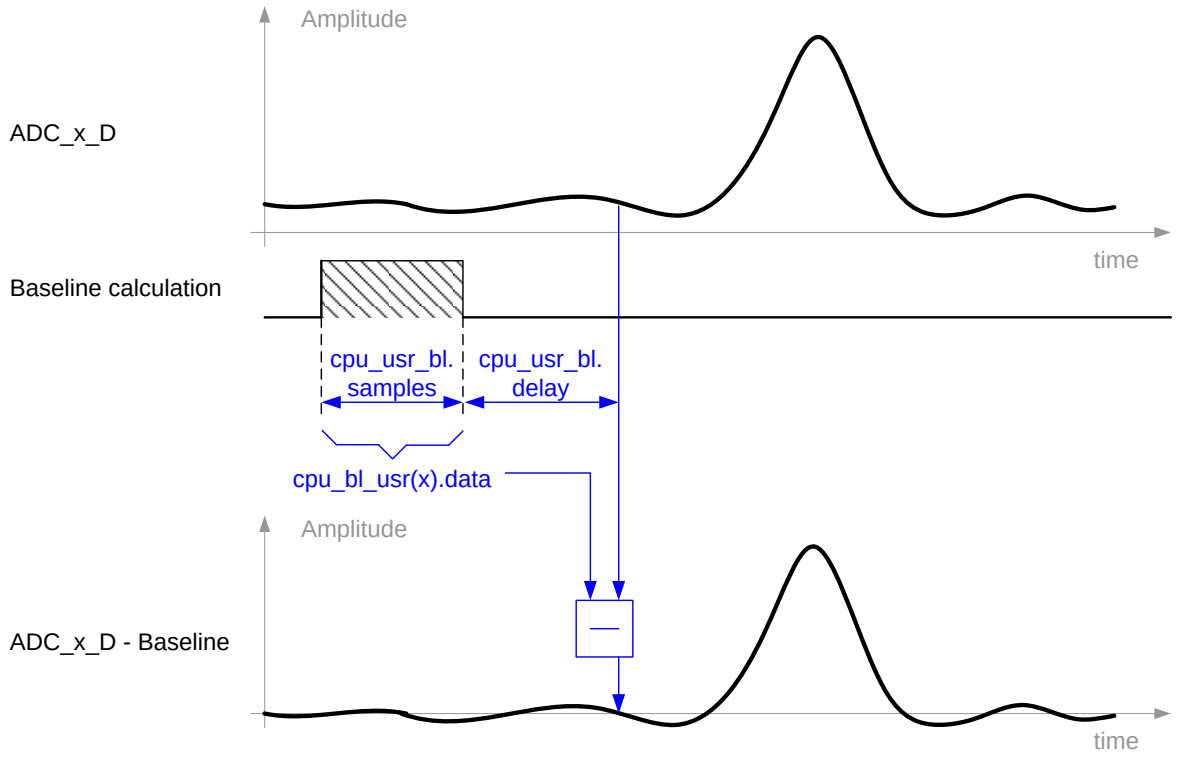

Figure 7: ADC_SMP_BASELINE Timing Diagram

The baseline is implemented as a configurable sliding window average and denoted as cpu_bl_usr(x).data that is available for the user by register access (for each ADC one register, hence x = 0 … 5). The baseline register consists of the baseline value (cpu_bl_usr(x).data) and a flag which indicates whether the baseline calculation has reached its number of samples (2^cpu_usr_bl.samples) for the sliding window. This is important because at startup the sliding window average has to start summing up the ADC samples and dividing them accordingly. For detecting wrong values at baseline startup, a flag (cpu_bl_usr(x).valid) is introduced signaling the validity of the baseline that can be read in a single read request (consistently = in the same 32 bit word) with the baseline data.

Address (REG_ADDR)	R/W	Bit	Name	Description
0x00000060	R/W	0	cpu_usr_bl. ena	Baseline enable '0' = Baseline calculation disabled '1' = Baseline calculation enables
0x00000064	R/W	3:0	cpu_usr_bl. samples	Baseline samples used for calculation set as 2^x 0 = 1 sample 1 = 2 samples 2 = 4 samples 3 = 8 samples 4 = 16 samples all other settings = 16 samples (same as setting 4)

0x00000068	R/W	9:0	cpu_usr_bl. delay	Baseline delay to the current measurement in number of clock cycles.
0x0000006C	R	15:0	cpu_bl_usr(0). data	Baseline calculated based on values from ADC 0.
		31	cpu_bl_usr(0). valid	Baseline ADC 0 valid.
0x00000070	R	15:0	cpu_bl_usr(1). data	Baseline calculated based on values from ADC 1.
		31	cpu_bl_usr(1). valid	Baseline ADC 1 valid.
0x00000074	R	15:0	cpu_bl_usr(2). data	Baseline calculated based on values from ADC 2.
		31	cpu_bl_usr(2). valid	Baseline ADC 2 valid.
0x00000078	R	15:0	cpu_bl_usr(3). data	Baseline calculated based on values from ADC 3.
		31	cpu_bl_usr(3). valid	Baseline ADC 3 valid.
0x0000007C	R	15:0	cpu_bl_usr(4). data	Baseline calculated based on values from ADC 4.
		31	cpu_bl_usr(4). valid	Baseline ADC 4 valid.
0x00000080	R	15:0	cpu_bl_usr(5). data	Baseline calculated based on values from ADC 5.
		31	cpu_bl_usr(5). valid	Baseline ADC 5 valid.

Table 3: ADC_SMP_BASELINE Register Map

4.5 ADC_SMP_BUFFER

For tests the sampled ADC data is stored consistently in a buffer. Depending on the samples requested (cpu_usr_fsm.s1_1) and the sampling mode (cpu_usr_fsm.mode) this marks the starting point of the stored ADC data that is made available to the user. The following figure is also shown in the chapter “ADC_SMP_FSM” and the registers are explained there in detail.

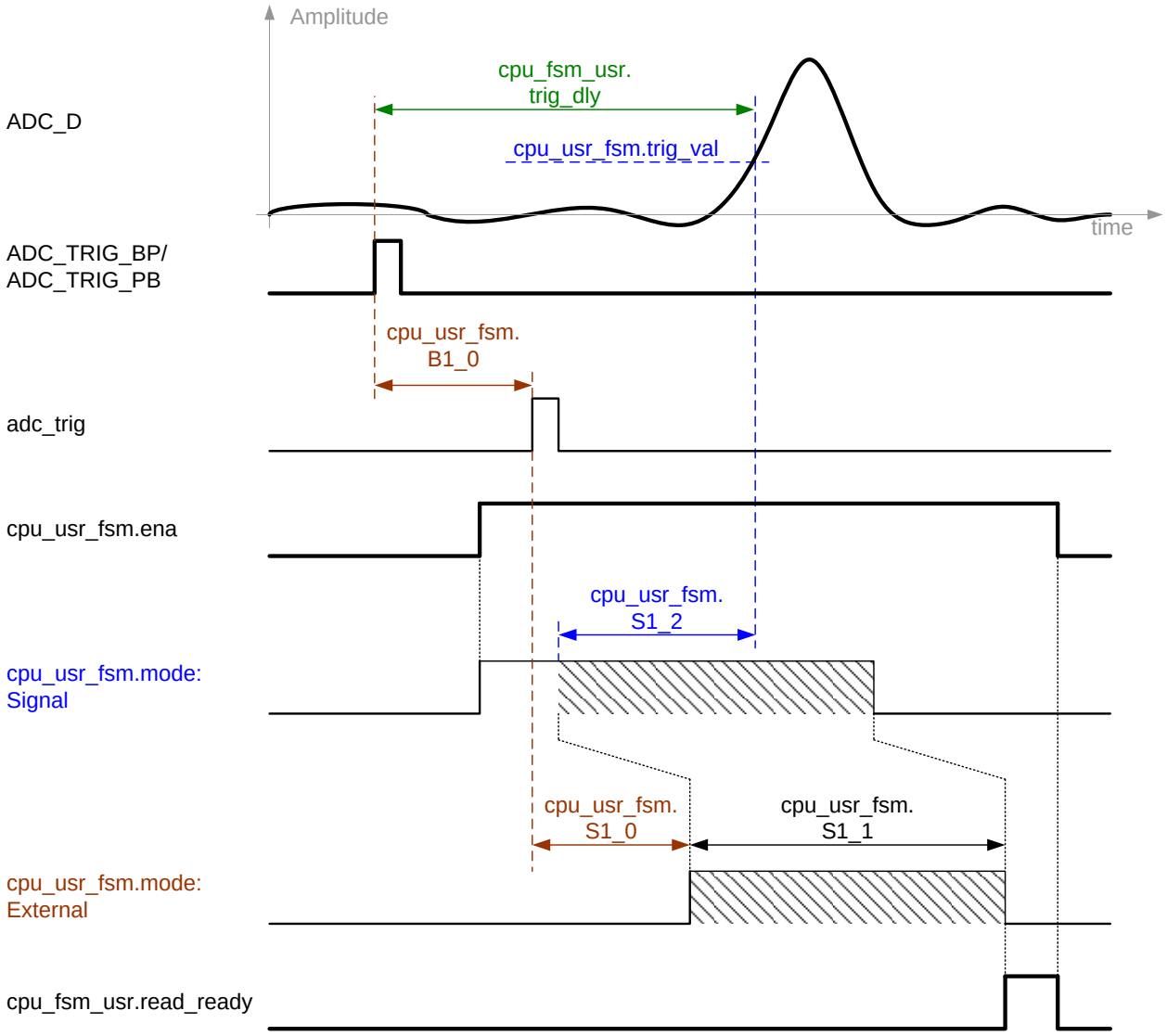

Figure 8: ADC_SMP_BUFFER Timing Diagram

The shaded area in the figure above denotes the data which is stored in the buffer until the user reads it or until the user flushes the buffer by disabling the measurement (cpu_usr_fsm.ena = ‘0’). A new measurement is started when the signal cpu_usr_fsm.ena is transitioning from ‘0’ to ‘1’. This behavior also allows slow IOCs to retrieve high speed signals consistently without mixing up old and new measurements.
The memory is set up to support single word and burst (DMA) transfers.
Please note: The size of the buffer is 1 kWord (32 bit words). If the user sets short sampling windows (cpu_usr_fsm.S1_1) then several sampling intervals will be stored in the buffer until it is full.

Address (MEM_ADDR)	R/W	Bit	Name	Description
0x000D0000: 0x000D0FFF	R	31:0	cpu_buffer_usr(0). data	Buffer ADC(0) data, measurement waveform data
0x000D8000: 0x000D8FFF	R	31:0	cpu_buffer_usr(1). data	Buffer ADC(1) data, measurement waveform data
0x000E0000: 0x000E0FFF	R	31:0	cpu_buffer_usr(2). data	Buffer ADC(2) data, measurement waveform data
0x000E8000: 0x000E8FFF	R	31:0	cpu_buffer_usr(3). data	Buffer ADC(3) data, measurement waveform data
0x000F0000: 0x000F0FFF	R	31:0	cpu_buffer_usr(4). data	Buffer ADC(4) data, measurement waveform data
0x000F8000: 0x000F8FFF	R	31:0	cpu_buffer_usr(5). data	Buffer ADC(5) data, measurement waveform data

Table 4: ADC_SMP_BUFFER Memory Map

4.6 ADC_SMP_CALC

The calculation block is the most complex part of the whole design; it is based on the document “Cavity BPM ADC Data Processing Ideas” [2].

First the finite state machine ADC_SMP_FSM defines the sampling window in which the peak of the reference I/Q (abs(r_ib)+abs(r_qb)) is recorded in the block MAX. For the block Interpolation, the I/Q reference sample before and the sample after the peak, has to be included in the peak detection measurement. The peak detection outputs are essentially the I/Q samples (x, y and r) when the reference channel has its peak. Please note: This is not true for the position channels when the beam passes on axis (0 offset), in such a case the position channels have an amplitude close to 0 and are dominated by a beam angle signal.

After finding the I/Q peaks, a conversion from cartesian to polar coordinates are calculated (in the block Cordic). The amplitudes (x_cd_amp, y_cd_amp and r_cd_amp) and phases (x_cd_phase, y_cd_phase and r_cd_phase) are stored for each bunch separately and can be retrieved by the control system. At the same time the Interpolation block calculates by means of a second order interpolation the top sample (s_top) and the deviation in degrees (t_top) from the peak sampled. This information is important in order to detect whether the sampling point is at the top of the I/Q analogue voltage pulse.

The next calculation steps are done to correct the various effects. First the I/Q imbalance has to be corrected. Because of differences in the I and the Q signal path the measurements of the amplitude and phase has to be adjusted. This is done by a lookup table correction as a function of the calculated Cordic phase (x_cd_phase, y_cd_phase and r_cd_phase).

The next correction accounts for the attenuator variation of amplitude and phase. Here as well a precalibrated table is used to do the correction based on the set attenuation (x_att, y_att and r_att).

The final correction accounts for the beam angle. The beam angle $\vec{B}$ is an orthogonal vector on the position vector $\vec{X}$ and $\vec{Y}$. The amplitude of this vector is proportional to the angle of the beam passing the cavity pickup.

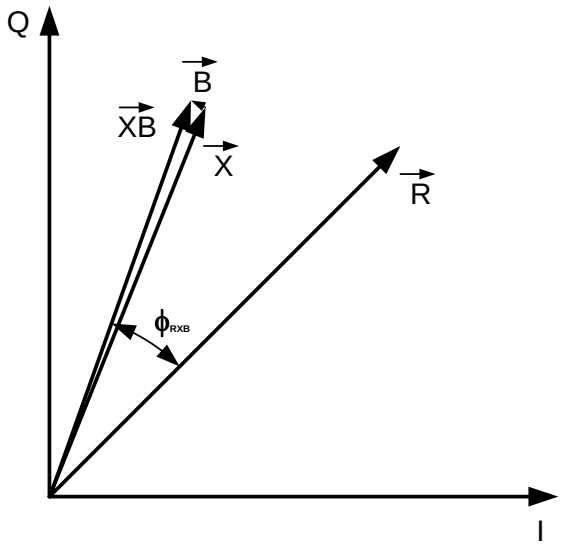

Figure 9: Beam Angle Correction

The problem of the beam angle is when the beam position is very close to the electrical zero because in such a case the beam angle vector dominates the position vector $\vec{XB} \approx \vec{B}$ and hence the position calculation. The result of the beam angle is not a clean zero crossing but a curved jump. The beam angle correction accounts for these effects. The correction works as follows: At a large position signal $\vec{XB} \approx \vec{X}$ the angle between the position and the reference channel is measured (x_ph_ref and y_ph_ref). For all following measurements only the projected vector to this large position reference is considered valid position information. Hence the correction factor is determined by first subtracting the current position angle from the current reference angle, then subtracting the position-reference angle at large position amplitudes, then the cosine of this angle is used as a correction factor for finally multiplying this cosine to the position vector amplitude.
The next step is to divide the pickup position information with the charge in order to obtain the charge independent position information followed by scaling (x_egu, y_egu q_egu) to physical units [mm and nC].
To enable comparison of position information for several pickups, the mechanical shifts (rotation of the pickup compared to an arbitrary reference) have to be calculated. This is carried out by a rotation matrix (rot11…rot22) before the samples are stored in user accessible buffers (Storage).

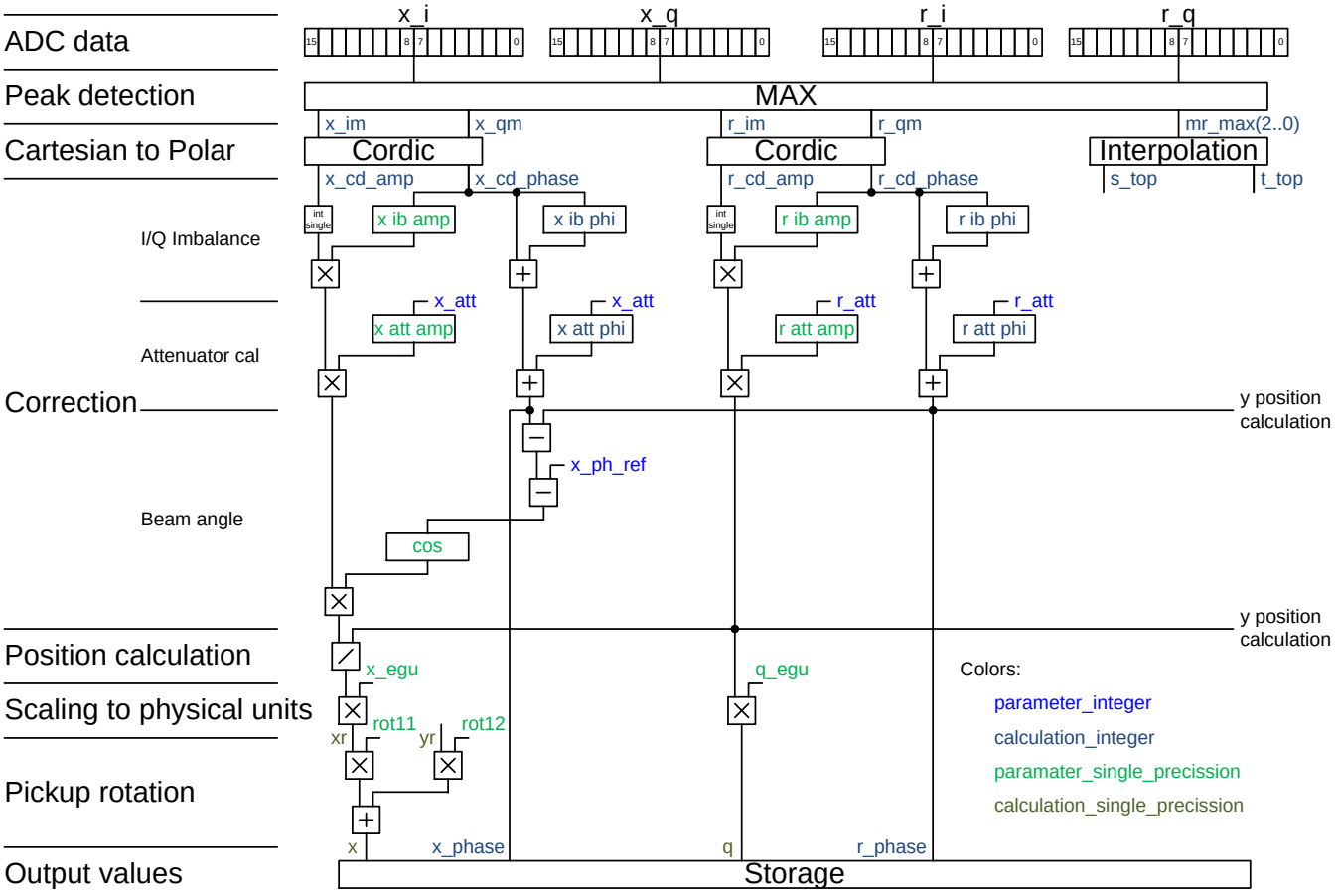

Figure 10: ADC_SMP_CALC Structure

Please note: The calculations above are not entirely executable within the shortest bunch spacing (B2.0) hence the calculation is implemented as a pipeline adding an initial latency; but delivering the calculation result with a rate of the bunch spacing (B2.0).

There is an important feature in the ADC_SMP_CALC block which is not described so far. The problem in question is: What should be presented to the user if no valid data is used? For example; the ADC samples are saturating at 32767 or –32768 (2complement 16 bit) or the charge is exactly 0 and hence the divider presented in the figure above calculates incorrect results (the same is true for each calculation step). The solution to this problem is implemented as a valid bit for each step in the pipeline. Each step of the calculation inherits the valid bit from the previous calculation and may set it to invalid if necessary, but an invalid bit from the predecessor remains invalid after the current block. For example: Let's assume the ADC samples are saturating, hence all calculation steps will inherit this information and propagate the invalid bit to the storage accessible to the user. Now let's assume the charge calculation is exactly 0, hence the position x and y will be invalid due to the division but all other calculated values are correct (although not very useful).
Therefore the user should remember to check (see chapter ADC_SMP_BRAM) the validity bit in addition to the position and charge information of each bunch.

Address (REG_ADDR)	R/W	Bit	Name	Description
0x00000090	R/W	0	cpu_usr_fsm. ib_ena	I/Q imbalance correction enable. '0' = I/Q calculation disabled. This means the amplitude is corrected by multiplying 1.0 and the phase by adding 0.0 °. '1' = I/Q calculation enabled. This means the amplitude is corrected by multiplying the value stored in the amplitude table and the phase by adding the value stored in the phase table.
0x000000A0	R/W	0	cpu_usr_fsm. ac_ena	Attenuation calibration enable. '0' = Attenuation calibration disabled. This means the amplitude is corrected by multiplying $10^{\frac{attenautor\_in\_dB-63}{20}}$ and the phase by adding 0.0 °. '1' = Attenuation calibration enabled. This means the amplitude is corrected by multiplying the value stored in the amplitude table and the phase by adding the value stored in the phase table.
0x000000A4	R/W	5:0	cpu_usr_fsm. ac_r	Attenuator set for reference channel.
0x000000A8	R/W	5:0	cpu_usr_fsm. ac_x	Attenuator set for position x channel
0x000000AC	R/W	5:0	cpu_usr_fsm. ac_y	Attenuator set for position y channel
0x000000B0	R/W	0	cpu_usr_fsm. ba_ena	Beam angle correction enable. '0' = Beam angle correction disabled. This means the amplitude is corrected by multiplying 1.0. '1' = Beam angle correction enabled. This means the amplitude is corrected by multiplying the cosine of the calculated projection angle.
0x000000B4	R/W	15:0	cpu_usr_fsm. x_ph_ref	Angle between reference channel and large x position channel amplitude. Fix_point representation 10.6 [°]
0x000000B8	R/W	15:0	cpu_usr_fsm. x_ph_ref	Angle between reference channel and large y position channel amplitude.

				Fix_point representation 10.6 [°]
0x000000C0	R/W	31:0	cpu_usr_fsm. rot11	Pickup rotation coefficient a11. IEEE 754 single precision representation.
0x000000C4	R/W	31:0	cpu_usr_fsm. rot12	Pickup rotation coefficient a12. IEEE 754 single precision representation.
0x000000C8	R/W	31:0	cpu_usr_fsm. rot21	Pickup rotation coefficient a21. IEEE 754 single precision representation.
0x000000CC	R/W	31:0	cpu_usr_fsm. rot22	Pickup rotation coefficient a22. IEEE 754 single precision representation.
0x000000D0	R/W	31:0	cpu_usr_fsm. q_egu	Scaling coefficient for the charge Q. IEEE 754 single precision representation.
0x000000E0	R/W	31:0	cpu_usr_fsm. x_egu	Scaling coefficient for the position X. IEEE 754 single precision representation.
0x000000F0	R/W	31:0	cpu_usr_fsm. y_egu	Scaling coefficient for the position Y. IEEE 754 single precision representation.

Table 5: ADC_SMP_CALC Register Map

4.7 ADC_SMP_BRAM

The calculated beam/bunch data is stored systematically in a buffer. Each calculated variable is stored consecutively in its own address range in order to make it accessible by means of burst (DMA) transfers by the control system.
The ADC_SMP_BRAM starts at each bunch train for each of the memory ranges individually at the lowest address and increments its position after storing the calculation result for this bunch. This means that although the calculation is implemented as pipeline, and therefore produces results at different points in time (calculation might even overlap in time), the relative offset from the lowest address is denoting exactly the same electron bunch. To summarize, the data stored in every offset of every calculation result buffer is of the same bunch. The maximal stored bunches is defined in ADC_SMP_FSM with the parameter cpu_usr_fsm.B0_0.

Address (MEM_ADDR)	**R/W**	**Bit**	**Name**	**Description**
0x00000000: 0x00003FFF	R/W	31:0	cpu_bram_usr. data(0)	Charge buffer Q data. Calculated charge for each bunch of the bunch-train. IEEE 754 single precision representation [nC]
0x00004000: 0x00007FFF	R	0	cpu_bram_usr. data(0)	Charge buffer Q data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00008000: 0x0000BFFF	R/W	31:0	cpu_bram_usr. data(1)	Position buffer X data. Calculated position X for each bunch of the bunch-train. IEEE 754 single precision representation [mm]
0x0000C000: 0x0000FFFF	R	0	cpu_bram_usr. data(1)	Position buffer X data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00010000: 0x00013FFF	R/W	31:0	cpu_bram_usr. data(2)	Position buffer Y data. Calculated position Y for each bunch of the bunch-train. IEEE 754 single precision representation [mm]
0x00014000: 0x00017FFF	R	0	cpu_bram_usr. data(2)	Position buffer Y data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid

0x00018000: 0x0001BFFF	R	31:0	cpu_bram_usr. data(3)	Calculation r_cd_phase buffer data. Calculated reference channel I/Q phase for each bunch of the bunch-train. Fix_point representation 10.6 [°]
0x0001C000: 0x0001FFFF	R	0	cpu_bram_usr. data(3)	Calculation r_cd_phase buffer data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00020000: 0x00023FFF	R	31:0	cpu_bram_usr. data(4)	Calculation r_cd_amp buffer data. Calculated reference channel I/Q amplitude for each bunch of the bunch-train. Signed integer ADC code.
0x00024000: 0x00027FFF	R	0	cpu_bram_usr. data(4)	Calculation r_cd_amp buffer data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00028000: 0x0002BFFF	R	31:0	cpu_bram_usr. data(5)	Calculation x_cd_phase buffer data. Calculated X channel I/Q phase for each bunch of the bunch-train. Fix_point representation 10.6 [°]
0x0002C000: 0x0002FFFF	R	0	cpu_bram_usr. data(5)	Calculation x_cd_phase buffer data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00030000: 0x00033FFF	R	31:0	cpu_bram_usr. data(6)	Calculation x_cd_amp buffer data. Calculated X channel I/Q amplitude for each bunch of the bunch-train. Signed integer ADC code.
0x00034000: 0x00037FFF	R	0	cpu_bram_usr. data(6)	Calculation x_cd_amp buffer data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00038000: 0x0003BFFF	R	31:0	cpu_bram_usr. data(7)	Calculation y_cd_phase buffer data. Calculated Y channel I/Q phase for each bunch of the bunch-train. Fix_point representation 10.6 [°]
0x0003C000: 0x0003FFFF	R	0	cpu_bram_usr. data(7)	Calculation y_cd_phase buffer data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00040000: 0x00043FFF	R	31:0	cpu_bram_usr. data(8)	Calculation y_cd_amp buffer data. Calculated Y channel I/Q amplitude for each bunch of the bunch-train. Signed integer ADC code.
0x00044000: 0x00047FFF	R	0	cpu_bram_usr. data(8)	Calculation y_cd_amp buffer data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00048000: 0x0004BFFF	R	31:0	cpu_bram_usr. data(9)	Calculation t_top buffer data. Calculated top sampling phase of the reference channel for each bunch of the bunch-train
0x0004C000: 0x0004FFFF	R	0	cpu_bram_usr. data(9)	Calculation t_top buffer data valid. Marks the calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00050000: 0x00053FFF	R	31:0	cpu_bram_usr. data(10)	Calculation s_top buffer data. Calculated top sampling amplitude of the reference channel for each bunch of the bunch-train
0x00054000:	R	0	cpu_bram_usr.	Calculation s_top buffer data valid. Marks the

0x00057FFF			data(10)	calculated data for each bunch of the bunch-train as valid or not. '0' = Not valid '1' = Valid
0x00058000: 0x0005FFFF	R/W	0	cpu_bram_usr. data(11)	I/Q imbalance coefficients for the R phase correction. Fix_point representation 10.6 [°]
0x00060000: 0x00067FFF	R/W	0	cpu_bram_usr. data(12)	I/Q imbalance coefficients for the R amplitude correction. IEEE 754 single precision representation.
0x00068000: 0x0006FFFF	R/W	0	cpu_bram_usr. data(13)	I/Q imbalance coefficients for the X phase correction. Fix_point representation 10.6 [°]
0x00070000: 0x00077FFF	R/W	0	cpu_bram_usr. data(14)	I/Q imbalance coefficients for the X amplitude correction. IEEE 754 single precision representation.
0x00078000: 0x0007FFFF	R/W	0	cpu_bram_usr. data(15)	I/Q imbalance coefficients for the Y phase correction. Fix_point representation 10.6 [°]
0x00080000: 0x00087FFF	R/W	0	cpu_bram_usr. data(16)	I/Q imbalance coefficients for the Y amplitude correction. IEEE 754 single precision representation.
0x00088000: 0x0008FFFF	R/W	0	cpu_bram_usr. data(17)	RFFE attenuator calibration for the R channel phase correction. Fix_point representation 10.6 [°]
0x00090000: 0x00097FFF	R/W	0	cpu_bram_usr. data(18)	RFFE attenuator calibration for the R channel amplitude correction. IEEE 754 single precision representation.
0x00098000: 0x0009FFFF	R/W	0	cpu_bram_usr. data(19)	RFFE attenuator calibration for the X channel phase correction. Fix_point representation 10.6 [°]
0x000A0000: 0x000A7FFF	R/W	0	cpu_bram_usr. data(20)	RFFE attenuator calibration for the X channel amplitude correction. IEEE 754 single precision representation.
0x000A8000: 0x000AFFFF	R/W	0	cpu_bram_usr. data(21)	RFFE attenuator calibration for the Y channel phase correction. Fix_point representation 10.6 [°]
0x000B0000: 0x000B7FFF	R/W	0	cpu_bram_usr. data(22)	RFFE attenuator calibration for the Y channel amplitude correction. IEEE 754 single precision representation.
0x000D0000: 0x000D7FFF	R	0	cpu_buffer_usr. data(0)	ADC raw value X channel I. 16bit signed integer ADC code.
0x000D8000: 0x000DFFFF	R	0	cpu_buffer_usr. data(1)	ADC raw value X channel Q. 16bit signed integer ADC code.
0x000E0000: 0x000E7FFF	R	0	cpu_buffer_usr. data(2)	ADC raw value R channel I. 16bit signed integer ADC code.
0x000E8000: 0x000EFFFF	R	0	cpu_buffer_usr. data(3)	ADC raw value R channel Q. 16bit signed integer ADC code.
0x000F0000: 0x000F7FFF	R	0	cpu_buffer_usr. data(4)	ADC raw value Y channel I. 16bit signed integer ADC code.
0x000F8000: 0x000FFFFF	R	0	cpu_buffer_usr. data(5)	ADC raw value Y channel Q. 16bit signed integer ADC code.

Table 6: ADC_SMP_BRAM Memory Map

5 Appendix

None.