

ADU Handler Module Description

	Organisatie / Organization	Datum / Date
Auteur(s) / Author(s): Eric Kooistra	ASTRON	
Controle / Checked: Gijs Schoonderbeek	ASTRON	
Goedkeuring / Approval: Andre Gunst	ASTRON	
Autorisatie / Authorisation: Handtekening / Signature Andre Gunst	ASTRON	

© ASTRON 2012
All rights are reserved. Reproduction in whole or in part is prohibited without written consent of the copyright owner.

Distribution list:

Group:	Others:
Andre Gunst (AG, ASTRON) Gijs Schoonderbeek (GS, ASTRON) Eric Kooistra (EK, ASTRON) Daniel van der Schuur (DS, ASTRON) Harm-Jan Pepping (HJP, ASTRON)	

Document history:

Revision	Date	Author	Modification / Change
0.1	2012-04-05	Eric Kooistra	Draft.
0.2	2013-02-1	Eric Kooistra	Improved the current solution of the ADUH for receiving the sampled data and discussed alternative solutions. Added overview of the Apertif subrack. Added description of the measurement set up. Added measurement results of the ADUH with the bn_capture design in the UniBoard back nodes in the Apertif subrack with 64 signal paths.

Table of contents:

1	Introduction.....	6
1.1	Purpose	6
1.2	Scope.....	6
1.3	Summary	6
1.3.1	ADUH firmware.....	6
1.3.2	Measurement setup.....	6
1.3.3	Correct data and stable timing.....	6
1.3.4	Data results.....	7
1.3.5	Timing results	7
2	Data input and timing between ADU and BN.....	8
2.1	System overview.....	8
2.2	Problem statement	9
2.3	Measurement setup.....	9
2.3.1	Hardware	9
2.3.2	Software.....	12
2.3.2.1	CW statistics	12
2.3.2.2	Data bit error detection using BIST	13
2.3.2.3	Data noise measurements.....	13
2.3.2.4	Timing statistics	18
2.4	Current solution	20
2.4.1	Hardware aspects.....	20
2.4.1.1	Correct data	20
2.4.1.2	Correct timing	20
2.4.2	Firmware aspects	20
2.4.2.1	Correct data	20
2.4.2.2	Correct timing	20
2.4.3	Implementation	21
2.4.3.1	Using IOE registers.....	21
2.4.3.2	Using IOE delay settings	21
2.4.3.3	Using PLL and clock tree.....	21
2.4.3.4	Correct data	21
2.4.3.5	Correct timing	22
2.4.3.6	Synthesis	24
2.4.4	Validation	25
2.4.4.1	Correct data	25
2.4.4.2	Correct timing	28
2.5	Alternative solutions	32
2.5.1	Hardware aspects.....	32
2.5.2	Firmware aspects	32
3	MMS_ADUH_QUAD	34
3.1	ADUH_QUAD	34
3.2	ADUH_QUAD_REG	34
3.3	Verification	35
4	ADUH_DD - ADC data receiver	36
4.1	ADUH_DD	36
4.1.1	Hardware interface	36
4.1.2	Design.....	37
4.1.3	Implementation	38
4.2	LVDSH_DD.....	39
4.2.1	Design.....	39
4.2.2	Verifcation.....	39

4.3	ADUH_PLL	39
4.4	LVDSH_PLL	39
5	ADUH_VERIFY - ADC test pattern verification	40
5.1	Hardware interface	40
5.2	Design.....	40
5.3	Implementation	41
5.4	Verifcation.....	41
5.5	Validation	41
6	MMS_ADUH_MONITOR.....	42
6.1	ADUH_MONITOR	42
6.2	ADUH_MONITOR_REG.....	42
7	Appendix : Data value measurement results using the BIST.....	44
8	Appendix : Data timing measurement results using a CW.....	46
8.1	SP 0, 1, 2, 3.....	46
8.2	SP 0, 1, 4, 5, 8, 9, 12, 13.....	47
8.3	SP 0, 2, 16, 18, 32, 34, 48, 50.....	49

Terminology:

ADC	Analogue Digital Converter
ADU	Analogue Digital Unit [2]
ADUH	ADU Handler firmware module
ADUH_DD	ADUH that uses DDIO input and no PLL to receive the samples
ADUH_PLL	ADUH that uses LVDS Rx with a PLL to receive the samples
ALM	Adaptive Logic Module (1 ALM in Stratix IV FPGA contains 20 FF)
BN	Back Node
CDR	Clock Data Recovery
CW	Carrier Wave
dclk	Digital clock (400 MHz double data rate clock from the ADC)
dp_clk	Data path clock (200 MHz data path processing clock)
DDIO	Double Data rate IO
DDR	Double Data Rate
DP	Data Path
FF	Flip Flop
FIFO	First In First Out
HDL	Hardware Description Language
IO	Input Output
IOE	IO Element
LCU	Local Control Unit
LSBit	Least Significant bit
LVDS	Low Voltage Differential Signalling
MM	Memory Mapped
MSBit	Most Significant bit
PAC	Power And Clock board
PLL	Phase Locked Loop
RF	Radio Frequency
sclk	Sample clock (800 MHz sample clock for the ADC)
SDR	Single Data Rate
SISO	Source in Sink Out
SNR	Signal Noise Ratio
SOSI	Source Out Sink In
TB	Test Bench
UNB	UniBoard
Wi	Word index

References:

1. "Detailed Design of the Digital Beamformer System for Apertif", ASTRON-RP-413, G. Schoonderbeek, A. Gunst, E. Kooistra
2. "ADU Board Design", ASTRON-RP-399, G. Schoonderbeek
3. "PAC Hardware Design Document", ASTRON-RP-453, G. Schoonderbeek
4. "BN Capture Design Description", ASTRON-RP-498, E. Kooistra, D. van der Schuur
5. "Specification for module interfaces using VHDL records", ASTRON-RP-380, E. Kooistra
6. "PPS Handler Module Description", ASTRON-RP-1374, E. Kooistra
7. "I2C SMBUS Module Description", ASTRON-RP-329, E. Kooistra
8. UNB = https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk/
9. UPE = \$UNB/Software/python, UniBoard Python Environment, see \$UPE/README.txt

1 Introduction

1.1 Purpose

This document describes the ADU Handler firmware module that contains VHDL components for receiving the ADC samples on the UniBoard back node (BN) from the Analogue Digital Unit (ADU) [2] that is used for Apertif [1]. The ADUH module also contains other VHDL components for verifying and monitoring the samples. The ADUH module is located at \$UNB/Firmware/modules/aduh [8].

1.2 Scope

The ADUH measurements with the bn_capture design have been done at 800 MSps for all 16 BN in an Apertif subrack. The measurements have only been done for certain temperatures of the ADC and the BN. The timing margins for correct sample data clocking and sample timing synchronization are sufficiently large though to expect that the ADUH will work reliably over the wider range of operational temperatures and also for higher sample rates (up to 1.6 Gbps, see section 2.4.4). The ADUH module can also be used for interfacing to other ADC boards that use the BN LVDS interface of UniBoard.

1.3 Summary

1.3.1 ADUH firmware

The ADUH module provides components for clocking in ADC samples via the BN LVDS inputs with correct data and stable timing. De ADUH firmware is described in sections 3, 4, 5 and 6. The ADUH is validated using the bn_capture design on the 16 BN of an Apertif subrack (Figure 1).

1.3.2 Measurement setup

The measurement setup consists of the Apertif subrack (section 2.3.1) and a set of Python scripts that can interface with the memory mapped (MM) peripherals on UniBoard and analyse the results (section 2.3.2). The Python scripts are located at \$UPE [9].

1.3.3 Correct data and stable timing

The stable sample timing solution (section 2.4) assumes that all UniBoard back nodes (BN) can use the same settings that are defined at synthesis. Therefore the solution relies on matched latencies for the distribution of the clock, pps and data from the central source to each BN. For the data the central source is a common RF source. For the timing the central source is a 10 MHz reference. On the BN the solution relies on using IOE registers (Figure 21) and PLLs, on the synchronous behaviour of clock trees in an FPGA and on the common_acapture instances with a small logic lock region (Figure 23) to have a stable data delay between two internal clock domains.

1.3.4 Data results

The sample data measurements showed that for all 64 signal paths (SP) in the Apertif subrack no sample data bit errors occurred on the ADU-BN interfaces (section 2.4.2.1 and appendix 7).

1.3.5 Timing results

The sample timing was measured for a subset of SP (section 2.3.1). The timing measurements (section 2.4.4.2) resulted in an optimum set of IOE delays and PLL phase settings of Table 2. The timing measurements showed that still occasionally a timing offset did occur (none for some SP and less than once in every 100 restarts for some other SP, see section 2.3.2.4 and appendix 8). The timing offsets are rare. Therefore if still necessary then investigating them further can be done when the *clk* traces to the BN0:3 on UniBoard have been matched as well (Figure 28).

2 Data input and timing between ADU and BN

2.1 System overview

Figure 1 shows a block diagram of the Apertif subrack that can input and process 64 analogue signal paths (SP). The analogue input signals are sampled by 8 analogue to digital units (ADU) and digital processed by 16 back nodes (BN) on 4 UniBoards (UNB). The 800 MHz sample clock (*sclk*) and 200 MHz digital processing clock (*clk*) are generated on the power and clock (PAC) board [3] by a PLL that is locked to an external reference 10 MHz clock. The 800 MHz sample clock (*sclk*) is used as a single data rate (SDR) clock. On the LVDS interface between ADU and BN the 8 bit samples arrive as double data rate (DDR) data together with a 400 MHz clock (*dclk*) that is divided from the 800 MHz sample clock by the ADC. The external timestamp reference is provided by a PPS pulse that is also locked to the 10 MHz. The interconnect between the 1 PAC, 4 UNB and 8 ADU is carried via a backplane PCB.

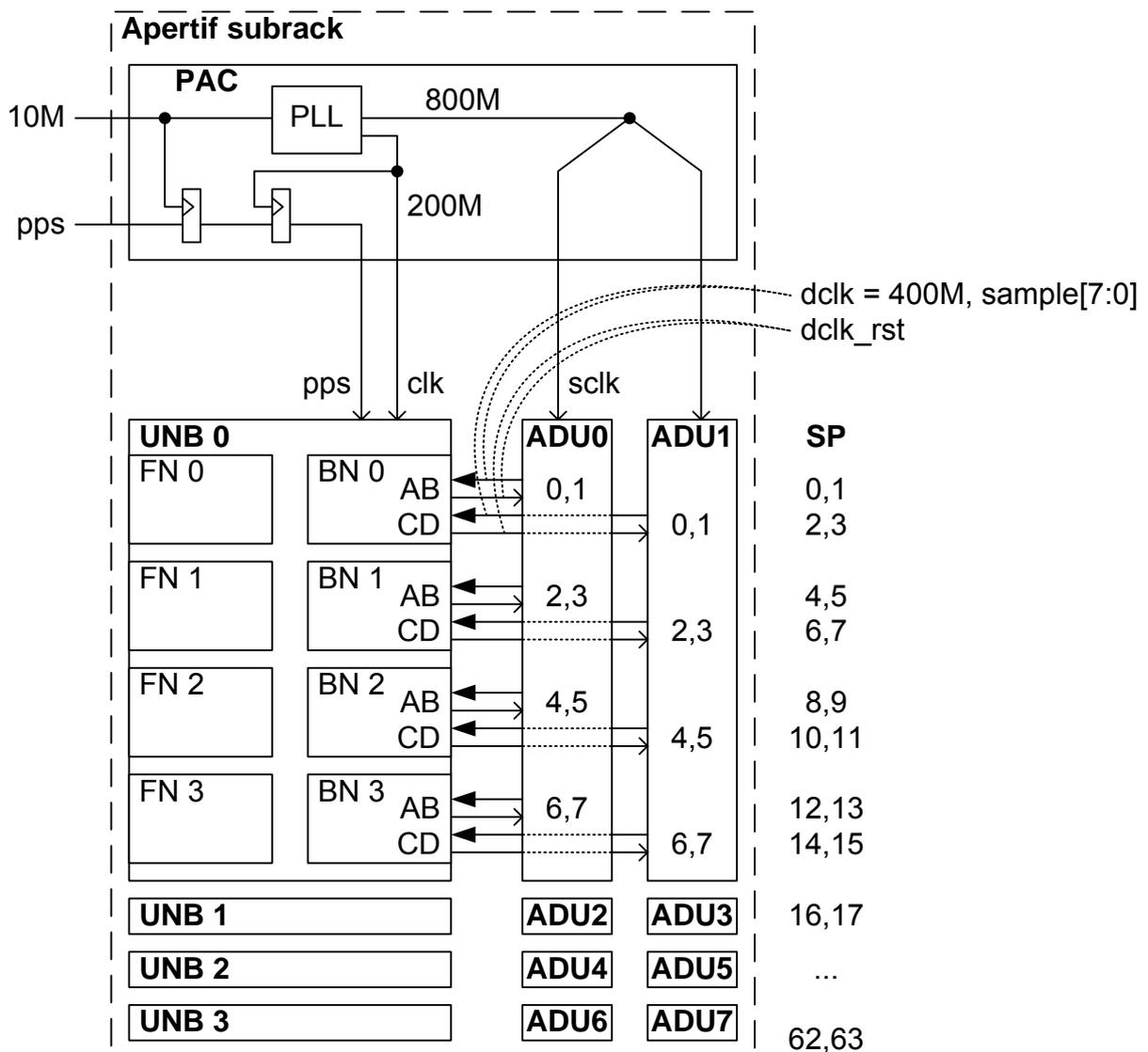


Figure 1: Apertif subrack for 64 analogue signal paths (SP)

2.2 Problem statement

The ADU Handler (ADUH) firmware is instantiated in each BN and can handle the interfacing of 4 signal paths from two ADU. Signal paths [A, B] from ADU [0] and signal paths [C, D] from ADU [1]. The ADU Handler has to receive the 800 MHz ADC samples across the LVDS interface between ADC and BN and capture them in the 200 MHz data path processing clock domain. It has to do this without data errors and with a fixed phase for the division from the 800 MHz clock to the 200 MHz clock. Inside the BN the ADUH outputs 4 ADC samples in parallel so packed per 32 bit word. The fixed phase for 1/4 clock division corresponds to a fixed phase for the 4x parallelization of the samples into data words.

The BN cannot process the samples directly at 800 MHz and even processing at 400 MHz makes it difficult to achieve timing closure on the FPGA logic. Therefore the BN *dp_clk* runs at 200 MHz and has to process 4 ADC samples in parallel. The *sclk* → *dclk* divide by 2 divider phase needs to be fixed because otherwise the samples from one ADU may be taken as data word [s0,s1,s2,s3] at timestamp 0 while for another ADU the data word at timestamp 0 can be shifted 1 sample e.g. [s1,s2,s3,s4]. Similar the *dclk* → *dp_clk* divide by 2 divider phase uncertainty can cause 2 samples shift uncertainty between two different BN - ADU interfaces.

The PPS also needs to be clocked into the *dp_clk* domain, because it is used to synchronize the start of the block sequence number (BSN) timestamp between the different BN. This capturing of the PPS is done by the PPS handler firmware module that is described in [6]. If the PPS is not clocked in at the same *dp_clk* cycle in all nodes then this shows as a 4 sample shift between two different BN - ADU interfaces.

2.3 Measurement setup

2.3.1 Hardware

Figure 2 shows the back side of the Apertif subrack where the 64 analogue SP can be applied to the 8 ADU. The ADU boards are numbered from 0 at the right to 7 at the left.

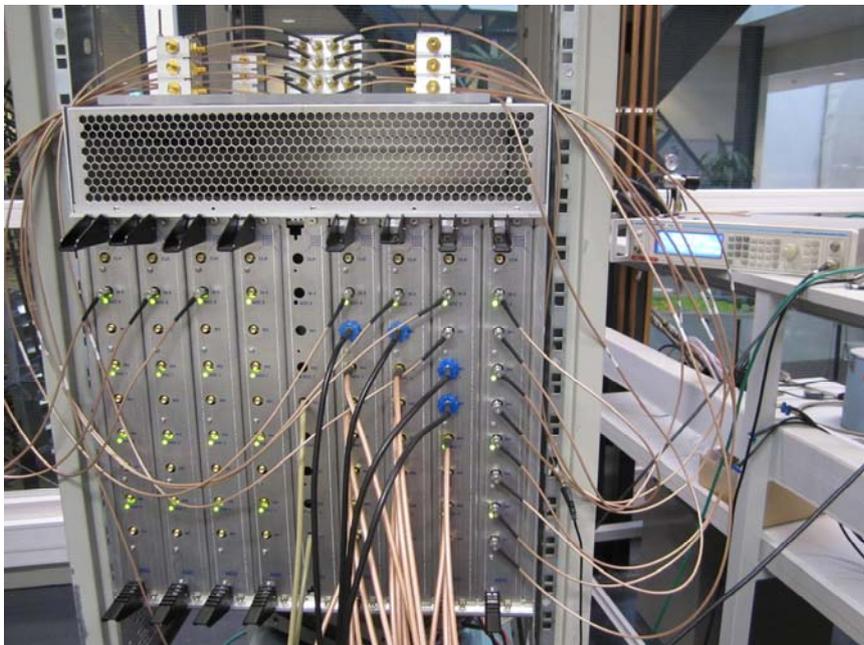


Figure 2: Photo of the back side of the Apertif subrack with 8 ADU

Figure 3 shows how the signal paths (SP) are numbered. For the BIST test pattern data measurements across the ADU-BN interface all 64 SP were measured. For the timing synchronization only the signal paths (SP) marked in gray in Figure 3 were measured. The top row of connectors connects from right to left to SP 0, 2, 16, 18, 32, 34, 48 and 50. The right column of connectors connects from top to bottom connects to SP 0, 1, 4, 5, 8, 9, 12, and 13. SP 3 is to include all SP of BN 0. This subset of SP covers all SP regarding location in the subrack and location on UniBoard. The RF signal splitter on top of the subrack in Figure 2 connects to the subset of SP that are marked in gray. The other SP were not measured for their analogue input because the RF splitter is not large enough and because they are expected to show similar behavior regarding the timing synchronization. The RF signal is an analogue CW that is generated by the signal generator that is visible at the right in Figure 2. All BN0:15 run the bn_capture firmware design [4].

BN 12		BN 8		BN 4		BN 0	
50	48	34	32	18	16	2	0
51	49	35	33	19	17	3	1
BN 15		BN 11		BN 7		BN 3	
54	52	38	36	22	20	6	4
55	53	39	37	23	21	7	5
58	56	42	40	26	24	10	8
59	57	43	41	27	25	11	9
62	60	46	44	30	28	14	12
63	61	47	45	31	29	15	13

Figure 3: Numbering of the SP on the back side of the Apertif subrack

Figure 4 shows the front side of the Apertif subrack with the PAC board in the middle and UniBoard 0, 1, 2 and 3 from left to right. The 10 MHz reference signal and PPS signal come from a FS725 Rubidium Frequency Standard that is visible in Figure 4 just below the subrack. The 10 MHz signal is also used to lock the RF signal generator so that the generated waveform is in lock with the clocks in the subrack.

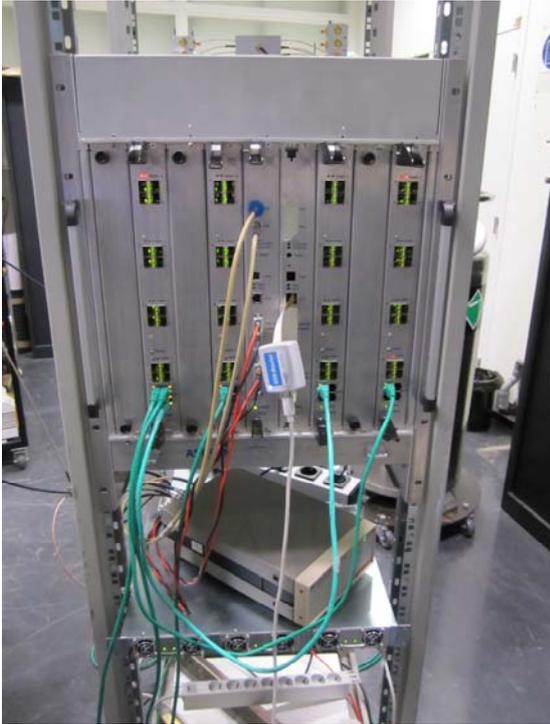


Figure 4: Photo of the front side of the Apertif subrack with PAC and 4 UNB

The RF signal generator is set to 700 MHz. After sampling at 800 MHz the captured signal appears as a 100 MHz sinus due to the under sampling as shown in Figure 5.

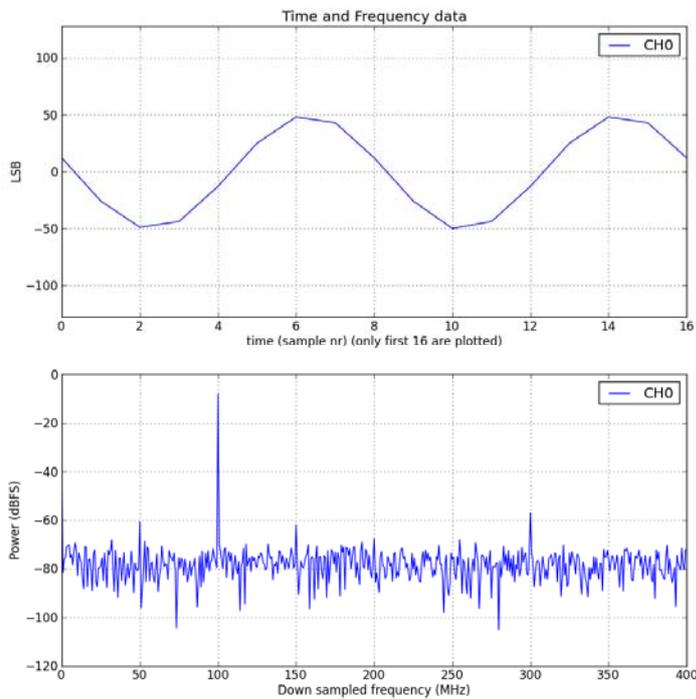


Figure 5: RF sinus captured in ADUH monitor buffer via SP 0

Figure 5 was obtained with Python script `tc_bn_capture_adc_plot.py` and shows the first 16 samples of the SP signals that are stored into the ADUH monitor buffer at every internal sync pulse. The internal sync pulse period is programmable to e.g. 200 M clock cycles. The periodic sync pulse is initiated at an external PPS pulse, so that the sync pulse in all nodes are aligned. The RF generator and the subrack clocks are locked to the external 10 MHz so therefore the view of is stable in time not only when ADU is kept in normal mode, but also when it is restarted in normal mode.

The ADCs on ADU can be configured via an I2C bus that is driven by the BN. The firmware supports some predefined I2C sequence that can be transferred using `i2c_commander` component [7]. The `Software/python/peripherals/pi_adu_i2c_commander.py` Python peripheral script provides methods for accessing the ADU via I2C and the `util_adu_i2c_commander.py` provides usage examples. For the measurements the ADC are set in test pattern mode to be able to run the BIST or in normal mode to be able to sample the analogue input signal.

The subrack is controlled by a Local Control Unit (LCU) via 1GbE. The LCU is a rack PC that is mounted below the subrack in Figure 4. The 1GbE switches on each UniBoard are used to interconnect all 4 UniBoard to the LCU. The FPGA images can be programmed via an USB blaster that connects to PAC (also shown in Figure 4) or the images can be stored in the flash and loaded into the FPGAs from there.

2.3.2 Software

Table 1 lists the scripts that are used for evaluating the ADUH in the `bn_capture` design. These scripts can in fact run with any design that has the `node_bn_capture` component instantiated. The ADUH in the `node_bn_capture` provides a set of memory mapped (MM) registers that can be controlled and monitored.

Python script	Description
<code>apps/adu_tests/tc_bn_capture_adc_plot.py</code>	Capture the ADC data for all signal paths
<code>base/ADC_functions.py</code>	Methods for calculating the timing statistics of the measured RF sinus.
<code>bn_capture/tc_bn_capture_adc_bist.py</code>	Set up and monitor the BIST for the ADC test pattern

Table 1: Python scripts for evaluating the ADUH with `bn_capture`

2.3.2.1 CW statistics

The assumption is that the analogue input is a CW signal with known period and that is locked to the sample clock. The zero-crossing of the measured CW in Figure 5 reveals the timing of the SP. The timing is synchronous for all SP if this zero-crossing occurs at the same time for all SP for every restart of the ADU (or power up of the subrack). There may be an offset between SP as long as this offset remains fixed. The zero-crossing instant can be observed manually but for intensive tests with $\gg 10$ ADU restarts and multiple SP it is necessary to automate the measurement. Therefore `tc_bn_capture_adc_plot.py` uses the method `add_clock_cw_statistics()` in `base/ADC_functions.py` to estimate the DC offset, phase φ and amplitude A of the captured CW. Furthermore the `base/ADC_functions.py` has methods to log and plot the various CW estimates.

The ADC samples are captured into the ADU Handler data buffer that can store 1024 samples per signal path. The buffering starts at the internal `sosi.sync` pulse [5] that is synchronous to the external PPS [6]. At CW frequency of 100 MHz the period $T=8$ samples so the ADUH monitor buffer can then store exactly 128 periods. First the DC level of the measured signal is determined and subtracted. Then the $CW(t)$ samples are multiplied by a reference $I(t) = \sin(\omega t)$ and $Q(t) = \cos(\omega t)$, where $\omega=2\pi/T$. Integrating the products yields two equations that solve phase φ and amplitude A . The zero-crossing of the RF sinus in Figure 5 occurs at about sample phase $\varphi = 4.3$. The statistics of φ for every time that the ADC on ADU is reprogrammed (restarted) into normal mode now reveal the stability of the timing distribution in the subrack. If for some ADU restarts there occur steps in φ of 1, 2, 3 or 4 sample times then these steps indicate synchronisation issues with respectively the `dclk_rst`, the FIFO reset, both `dclk_rst` and FIFO reset, or with the PPS. When the zero-crossings do not jump one or more integer sample periods then this means that the $sclk \rightarrow dclk \rightarrow dp_clk$

dividers divide with fixed phase relative to the *dp_clk* and that the PPS has started the *sync* at the same *dp_clk* cycle in all BN.

With the estimated the DC level, phase φ and amplitude A of the captured RF wave the noise level can be estimated by subtracting the estimated CW from the measured CW. This yields the noise signal $N(t)$. Any large peaks in the noise sample values indicate bit errors. For the noise signal $N(t)$ the noise power *noisePower* is calculated. The power of a full-scale CW is $cwPower = A^2/2$ where $A = 127.5$ for the 8-bit ADC. The estimated SNR = $10 * \log_{10}(cwPower / noisePower)$. The theoretical SNR for an 8 bit ADC is $6.02 * 8 \text{ bit} + 1.76 = 50 \text{ dB}$.

2.3.2.2 Data bit error detection using BIST

The ADC → BN LVDS data interface is validated for bit errors by checking that the received data matches exactly the test pattern that the ADC outputs in test mode (using *tc_bn_capture_adc_bist.py*). The BIST test can run for hours or days, because it is evaluated continuously in *aduh_verify*.

2.3.2.3 Data noise measurements

The CW measurements input (using *tc_bn_capture_adc_plot.py*) show that the noise peak values remained within < 2 units for all measurements and on average the noise peak values were about 1.3 units. This implies that no bit errors occurred for bit [6:1] and probably no bit errors occurred for bit [0]. Bit [7] was not covered, because with 13 dBm RF signal level at the input of the RF 1:32 splitter the amplitude of the measured CW was about 51 units (see Figure 5). The measured SNR were between about 45 and 53 dB across all SP. The noise level test can only measure the sampled data for the 1024 samples that are captured in the ADUH monitor buffer, so it is only nice to have compared to the BIST.

The following figures show the measured DC levels, amplitudes, noise peaks and SNR results for SP that are marked in gray in Figure 3. The results are plotted for 3000 ADU restarts using a *bn_capture* image that included the DDR3 logic and used the settings of Table 2. The results were obtained by means of running:

```
> python apps/adu_tests/tc_bn_capture_adc_plot.py --unb 0 --bn 0 --sp 0,1,2,3 --rep 3000 -v 3 -s <bn0>
> python apps/adu_tests/tc_bn_capture_adc_plot.py --unb 0 --bn 0:3 --sp 0,1 --rep 3000 -v 3 -s <column>
> python apps/adu_tests/tc_bn_capture_adc_plot.py --unb 0:3 --bn 0 --sp 0,2 --rep 3000 -v 3 -s <row>
```

Note that SP-0 is included in all three runs, so it gets tested 9000 times. The detailed log results that correspond to the plots are listed in appendix 8.

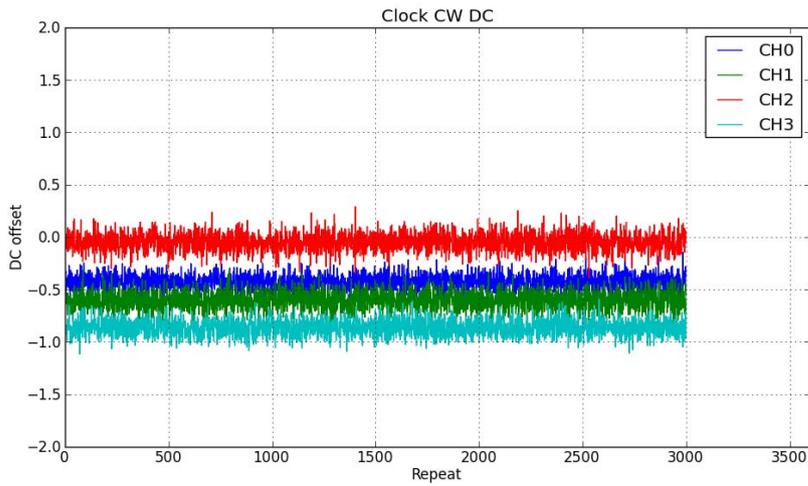


Figure 6: clock_cw_dcs_ch_0_1_2_3_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

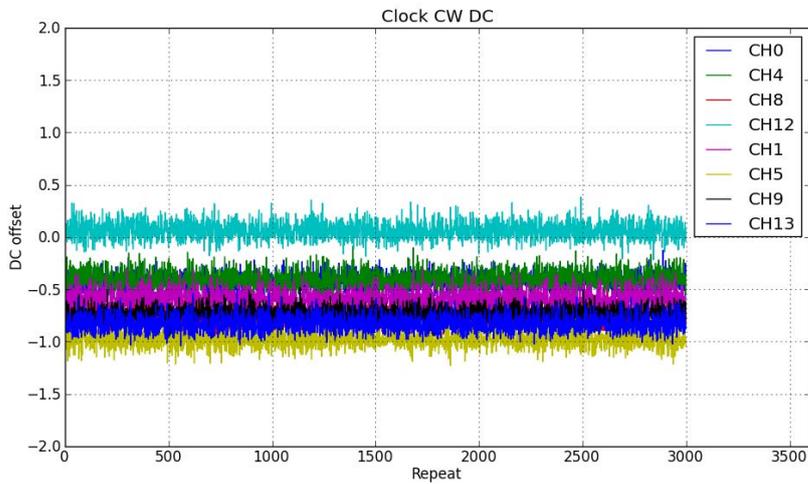


Figure 7: clock_cw_dcs_ch_0_1_4_5_8_9_12_13_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

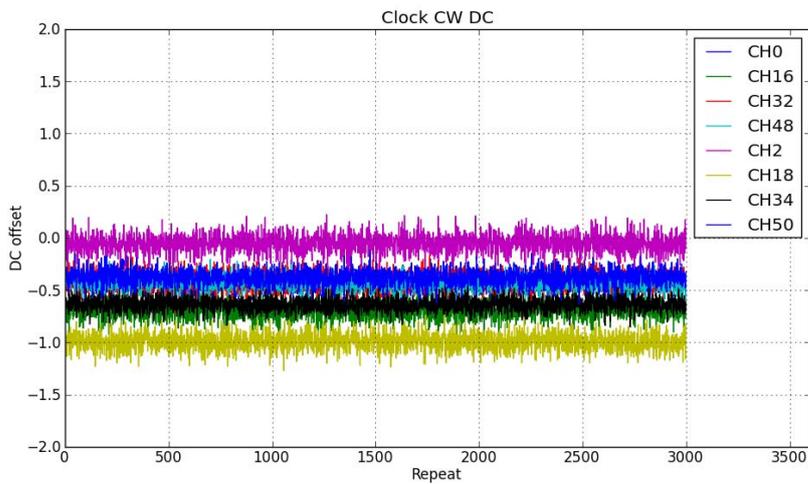


Figure 8: clock_cw_dcs_ch_0_2_16_18_32_34_48_50_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

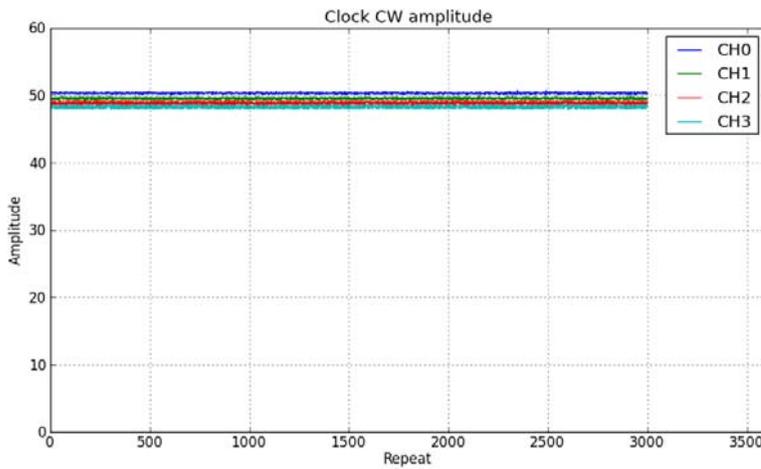


Figure 9: `clock_cw_amplitudes_ch_0_1_2_3_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0`

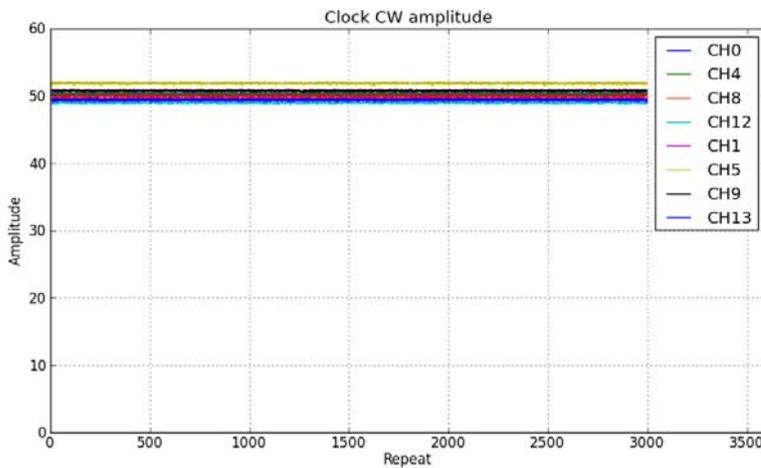


Figure 10: `clock_cw_amplitudes_ch_0_1_4_5_8_9_12_13_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0`

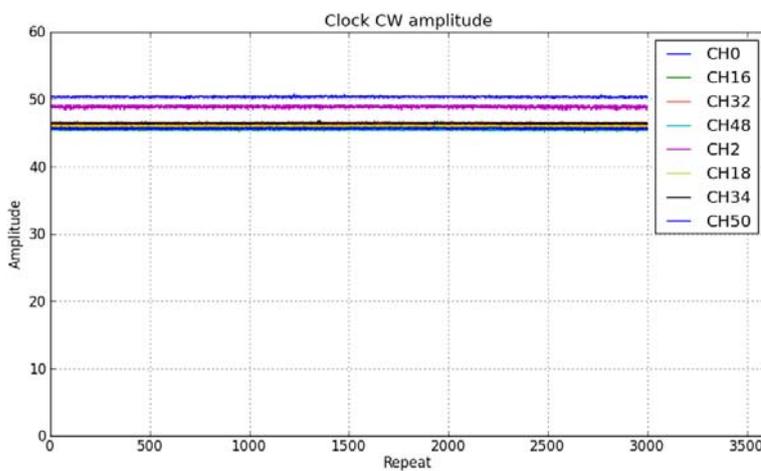


Figure 11: `clock_cw_amplitudes_ch_0_2_16_18_32_34_48_50_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0`

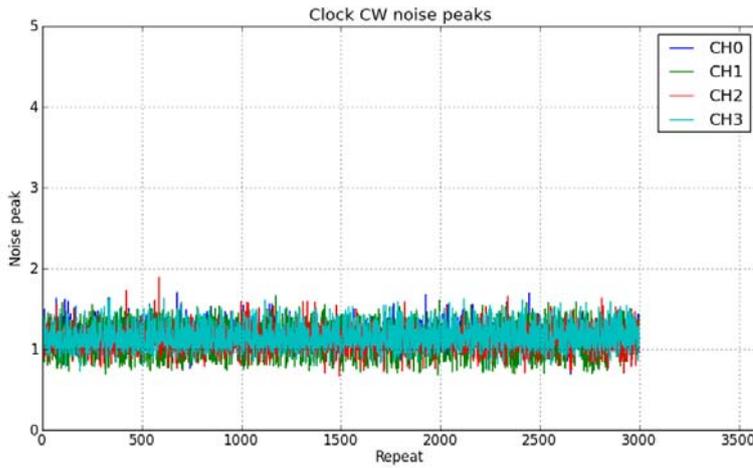


Figure 12: `clock_cw_noise_peaks_ch_0_1_2_3_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0`

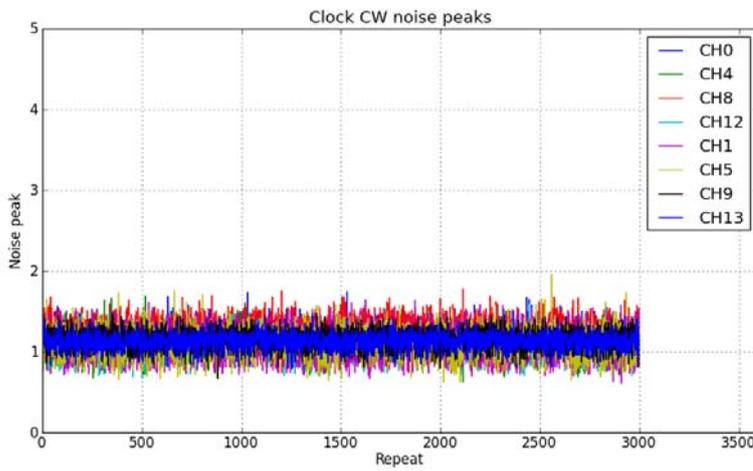


Figure 13: `clock_cw_noise_peaks_ch_0_1_4_5_8_9_12_13_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0`

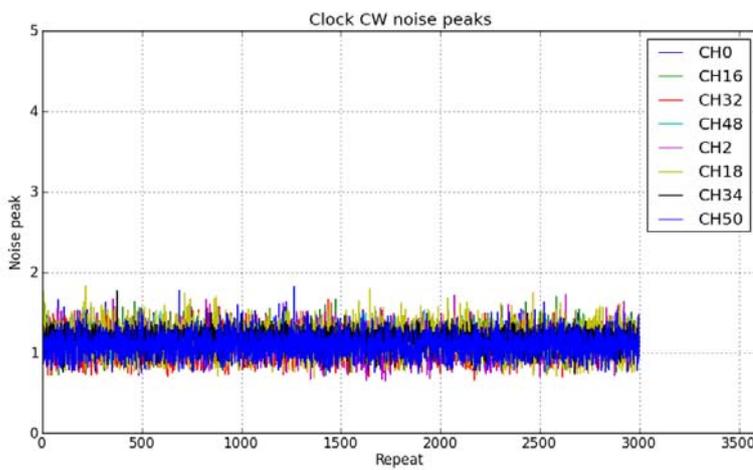


Figure 14: `clock_cw_noise_peaks_ch_0_2_16_18_32_34_48_50_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0`

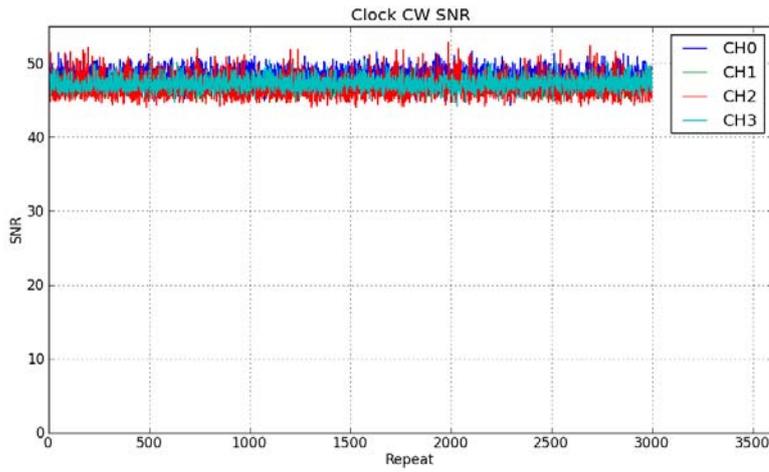


Figure 15: clock_cw_snrs_ch_0_1_2_3_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

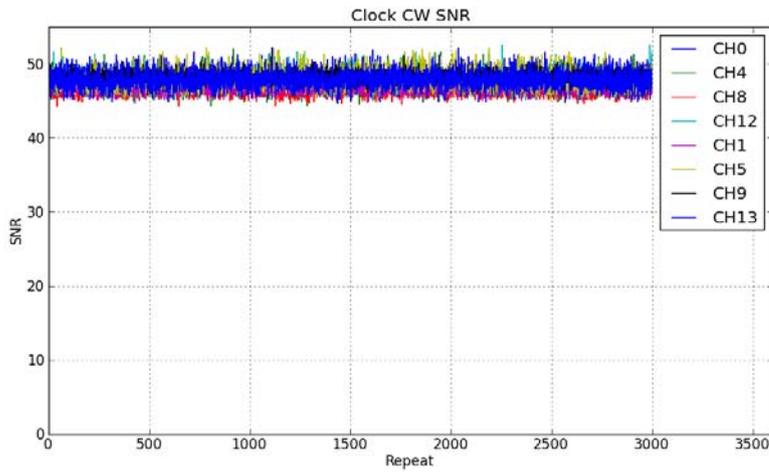


Figure 16: clock_cw_snrs_ch_0_1_4_5_8_9_12_13_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

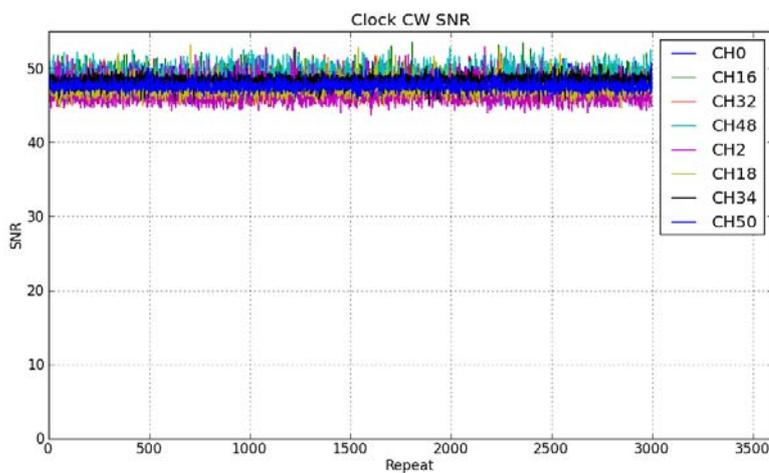


Figure 17: clock_cw_snrs_ch_0_2_16_18_32_34_48_50_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

2.3.2.4 Timing statistics

The following figures show the phase estimates for 3000 restarts of ADU. The phase of SP 0 \approx 4.32 for all runs, similar as in Figure 5. The phases of the other SP differ compared to the phase SP 0 but are almost constant too as they should be.

From Figure 19 and Figure 20 and the details in appendix 8.2 and 8.3 it shows that for SP 0,2 on BN0, SP 4,5 on BN1, and SP 12,13 on BN3 a few times a 4 sample timing offset occurred. It is unclear why these timing offsets occur. It is unlikely that they are due to the PPS-*dp_clk* synchronization. From Figure 27 it is expected that also with PPS input delay D3=0 the PPS is clocked in stable with almost maximum timing margin with respect to the rising edge of the *dp_clk*.

In another test using the same settings for *bn_capture*, but now without DDR3 logic. The phase of SP 0 \approx 4.32 so this shows that the timing was independent of the amount of other logic in the *bn_capture* design. The other results were that in 17500 restarts SP 12,13 showed four times a 4 sample timing offset and two times a 2 sample timing offset. All other SP then showed no timing offsets.

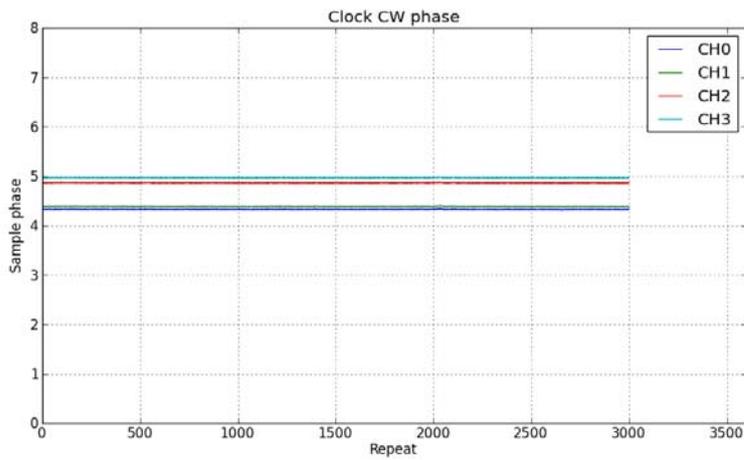


Figure 18: clock_cw_phases_ch_0_1_2_3_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

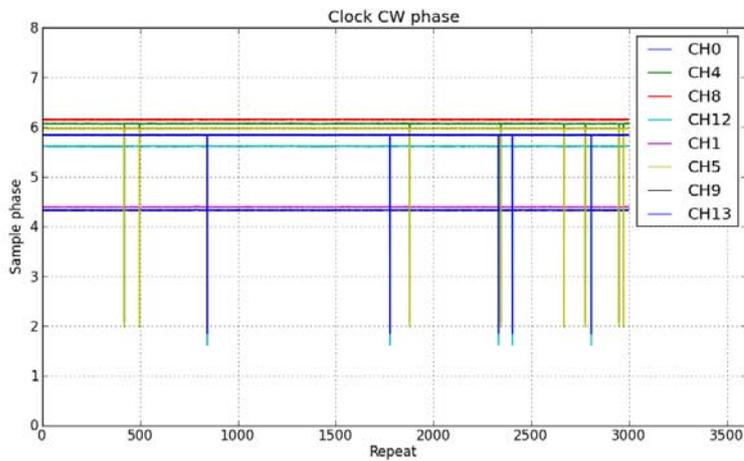


Figure 19: clock_cw_phases_ch_0_1_4_5_8_9_12_13_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

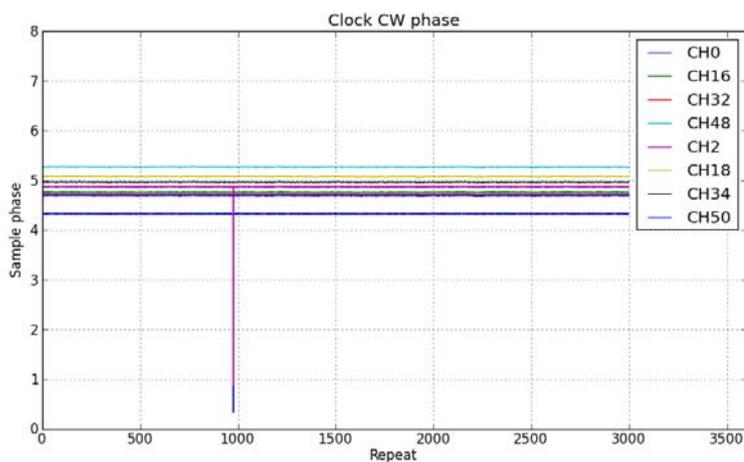


Figure 20: clock_cw_phases_ch_0_2_16_18_32_34_48_50_repeat_3000_ddr_phs_11_25_pps_0_clkrst_0

2.4 Current solution

2.4.1 Hardware aspects

2.4.1.1 Correct data

The ADC outputs the sample data to the BN together with a 400 MHz *dclk* clock. The BN can use this *dclk* to reliably clock in the sample data into the FPGA.

2.4.1.2 Correct timing

The clock distribution traces, the pps traces and the sample data traces on the boards and on the backplane are all matched in length to ensure that at each BN corresponding signals arrive at the same global time (within a margin that is sufficiently small compared to the sample clock period of 1250 ps). In this way the same firmware image can run on all BN.

The phase of the divide by 2 clock divider in the ADC on ADU that output the 400 MHz *dclk* can be reset to have a using the *dclk_rst* signal from the 200 MHz *dp_clk* domain on the BN FPGA to the 800 MHz *sclk* sample clock domain on ADU.

2.4.2 Firmware aspects

2.4.2.1 Correct data

The current *mms_aduh_quad* VHDL module described in section 3 uses the *aduh_dd* component that relies on using the input delay elements D1, D2, and D3 in the IOE at the FPGA pin to ensure that the ADC data is stable when it is clocked in. The *aduh_dd* component uses two *lvdsh_dd* components to receive the data from two ADUs. See Figure 31 and Figure 32 for a block diagram of the *aduh_dd* and *lvdsh_dd* components.

2.4.2.2 Correct timing

The phase of the 800M *sclk* sample clock → 400 MHz *dclk* clock divider in the ADC can be set reliably using the *dclk_rst* signal. The clock divider in the ADC will start at a random phase, causing 0 or 1 *sclk* clock cycle phase uncertainty in the starting phase of the *dclk*. This then shows as a 0 or 1 sample offset between the inputs from two different ADCs. The starting phase of the clock divider in the ADC can be set with a pulse from the 200 MHz *dp_clk* clock domain onto the *dclk_rst* BN output to ADC input pin. In the VHDL the 200 MHz clock domain is referred to as the *dp_clk* (DP = Data Path) clock domain.

The clock division from the 400 MHz *dclk* domain to the 200 MHz *dp_clk* domain is taken care of using a mixed width FIFO (see Figure 32). While the *dclk_rst* pulse is issued, the FIFO in the BN that transfers the data from the 400 MHz *dclk* domain to the 200 MHz *dp_clk* domain is reset as well. This causes the FIFO to become empty. This FIFO is a mixed width FIFO, which means that for every two 16 bit words that are written there can be read one 32 bit word. The 400 MHz *dclk* writes the sample for two SP input streams in parallel and the 200 MHz *dp_clk* reads them out with two samples for these two SP input streams in parallel.

During the *dclk_rst* pulse the *dclk* stops and after the *dclk_rst* release the *dclk* starts again, but then with a defined divider phase. The FIFO was empty so the first data that is then written by the *dclk* is also the first 16 bit word in the FIFO, therefore the phase of the 400 MHz *dclk* to the 200 MHz *dp_clk* is thereby then also defined. In summary:

1. The *dclk_rst* pulse resets the phase of the 800M sample *sclk* → 400M *dclk* divider with respect to the *dp_clk*
2. The FIFO reset resets the phase of the 400M *dclk* → 200M *dp_clk* division with respect to the *dp_clk*

The 800 MHz *sclk* sample clock and the 200 MHz *dp_clk* are in lock thanks to the Power and Clock (PAC) board, see Figure 1. This implies that during normal operation the FIFO can not overflow or run empty. The *common_fifo_dc_lock_control* component in Figure 32 lets the FIFO get filled to a certain level and from then on it enables the reading of the FIFO. As long as the filling level of the FIFO remains the same then that is

reported as ADUH-locked, see section 3.2. When the FIFO does run empty (or overflows) then there is something wrong between the *dclk* and the *dp_clk* and this then gets reported as ADUH-not-locked. The *common_fifo_dc_lock_control* continuously tries to regain lock by restarting again.

The fact that the 800 MHz *sclk* sample clock and the 200 MHz *dp_clk* are in lock also implies that the *dclk_rst* pulse can be passed on reliably from the 200 MHz *dp_clk* domain into the 800 MHz *sclk* sample clock domain, provided that the *dclk_rst* signal has sufficient setup and hold margin with respect to the rising edge of the 800 MHz *sclk* sample clock near the ADC on ADU.

2.4.3 Implementation

2.4.3.1 Using IOE registers

The IOE registers are close to the pin so by using these registers e.g. via DDIO the IO timing is guaranteed to be the same independent of how much logic the rest of the design has.

2.4.3.2 Using IOE delay settings

If possible the IOE delay values should be chosen as close to 0 as possible, because then any temperature or chip process dependency of the delay setting has the least impact.

If possible it is better to delay a clock than to delay a data bus, because there will be a mismatch between the delay elements of the individual data lines.

It seems not possible to control the IOE delay elements dynamically via the MM interface, because not all IO pins used for the BN-ADU LVDS interface support the configuration clock that is needed for that. Therefore the IOE delay settings are fixed after synthesis.

With *aduh_dd_top.vhd* and *aduh_dd.sdc* it was found that the IOE input delay elements can be invoked using 'set_input_delay' in the SDC file. However it seems not possible to invoke the IOE output delay elements using 'set_output_delay'. Instead these output delays can directly be set by via a pin assignment in the pinning TCL file. Using pin assignment to set the IOE delays is easier than to use timing constraints, therefore all IOE delays are set via the pinning TCL files.

2.4.3.3 Using PLL and clock tree

The PLL in *ctrl_unb_common* is used in "NORMAL" mode and ensures that the *dp_clk* inside the FPGA has a fixed phase offset (delay) with respect to the 200 MHz clock that arrives at the FPGA pin. The clock tree distribution network in the FPGA guarantees that the *dp_clk* arrives at every flip flop (FF) with that same delay.

Using a PLL to adjust the output phase of a clock is preferred over using IOE delays, because the PLL phase setting is independent of temperature and chip process variations.

2.4.3.4 Correct data

Figure 21 shows the IOE for the sample data input in the BN FPGA. The D3 input delay element is used for each data bit to delay the data with respect to the *dclk*. The *dclk* itself cannot be delayed, because it is a global clock and the place and route then gives an error, therefore the data bus needs to be delayed instead. The D3 can be set per sample input pin. For example as done in `$UNB\designs\unb_common\src\tcl\BACK_NODE_adc_pins.tcl`:

```

set_instance_assignment -name D1_DELAY 0 -to ADC_BI_A[0]
set_instance_assignment -name D2_DELAY 0 -to ADC_BI_A[0]
set_instance_assignment -name D3_DELAY 2 -to ADC_BI_A[0]

set_instance_assignment -name D1_DELAY 0 -to ADC_BI_C[0]
set_instance_assignment -name D2_DELAY 0 -to ADC_BI_C[0]
set_instance_assignment -name D3_DELAY 3 -to ADC_BI_C[0]

```

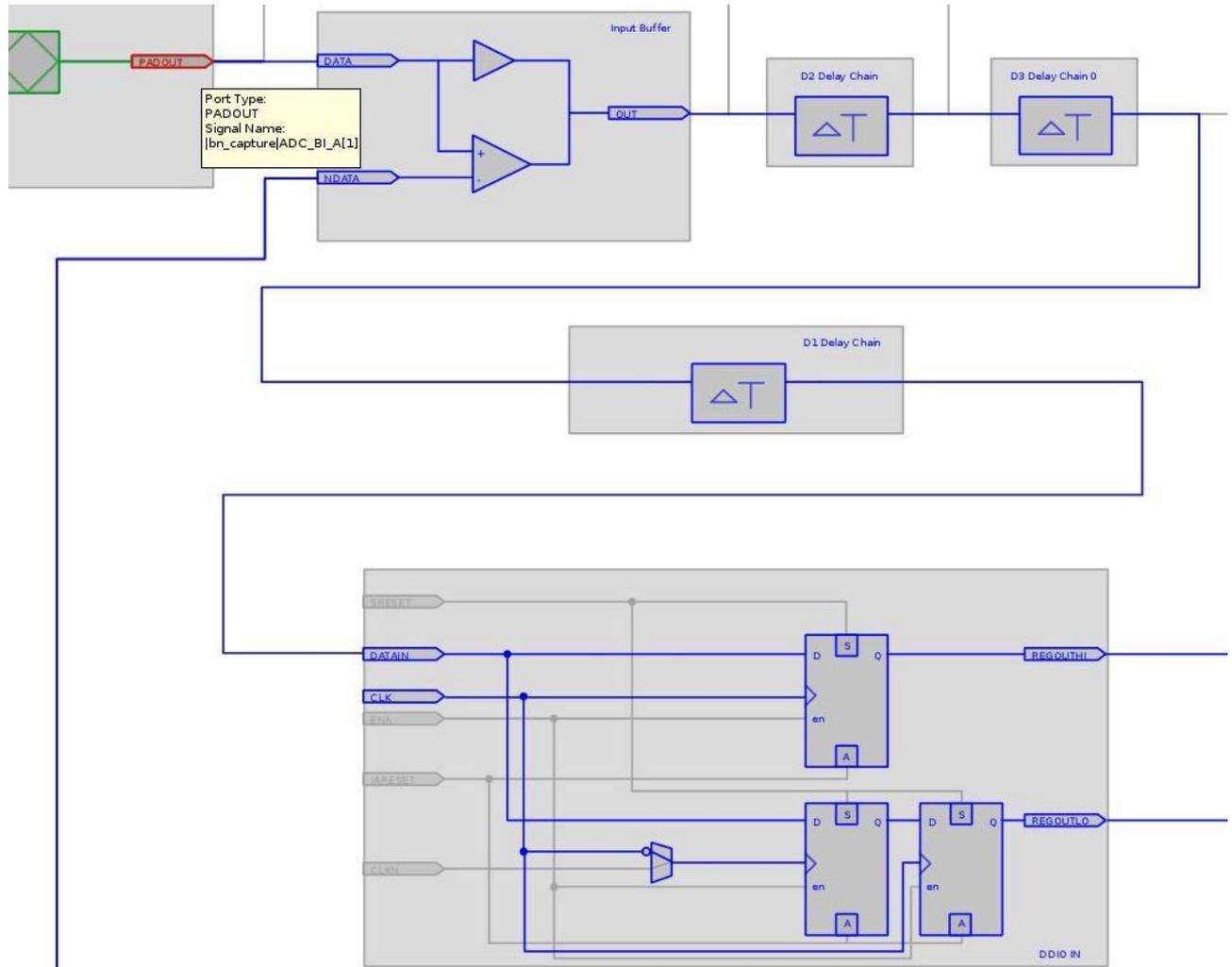


Figure 21: Input for ADC_BI_A[1] using D3 and DDIO

2.4.3.5 Correct timing

For the *dclk_rst* the requirement is that it must be issued from the 200 MHz *dp_clk* clock domain. The *dp_clk* is driven by a PLL in *ctrl_unb_common* and its phase can be adjusted such that the timing of *dclk_rst* suits the setup and hold requirements of the *sclk* in at the ADC on ADU. It is not necessary to add another clock output to this PLL with the appropriate offset phase for *dclk_rst*, because the rest of the functionality that uses the *dp_clk* works fine this phase offset (including the capture of the PPS). Using only one PLL clock output saves the (power) cost of using an extra clock tree resource in the FPGA. The phase of the *dp_clk* is set via *g_dp_clk_phase* = "156" ps in *ctrl_unb_common* for 11.25 degrees at 200 MHz.

Figure 22 shows the IOE with the output delay elements D5 and D6 to adjust the *dclk_rst* signal for the setup and hold requirements of the *sclk* in at the ADC on ADU. Both can be set to 0, because the PLL phase offset

for the *dp_clk* already provides sufficient control. The IOE delays can be set for example as done in `UNB\designs\unb_common\src\tcl\BACK_NODE_adc_pins.tcl`:

```
set_instance_assignment -name D5_DELAY 0 -to ADC_BI_A_CLK_RST
set_instance_assignment -name D5_DELAY 0 -to ADC_BI_D_CLK_RST
set_instance_assignment -name D6_DELAY 0 -to ADC_BI_A_CLK_RST
set_instance_assignment -name D6_DELAY 0 -to ADC_BI_D_CLK_RST
```

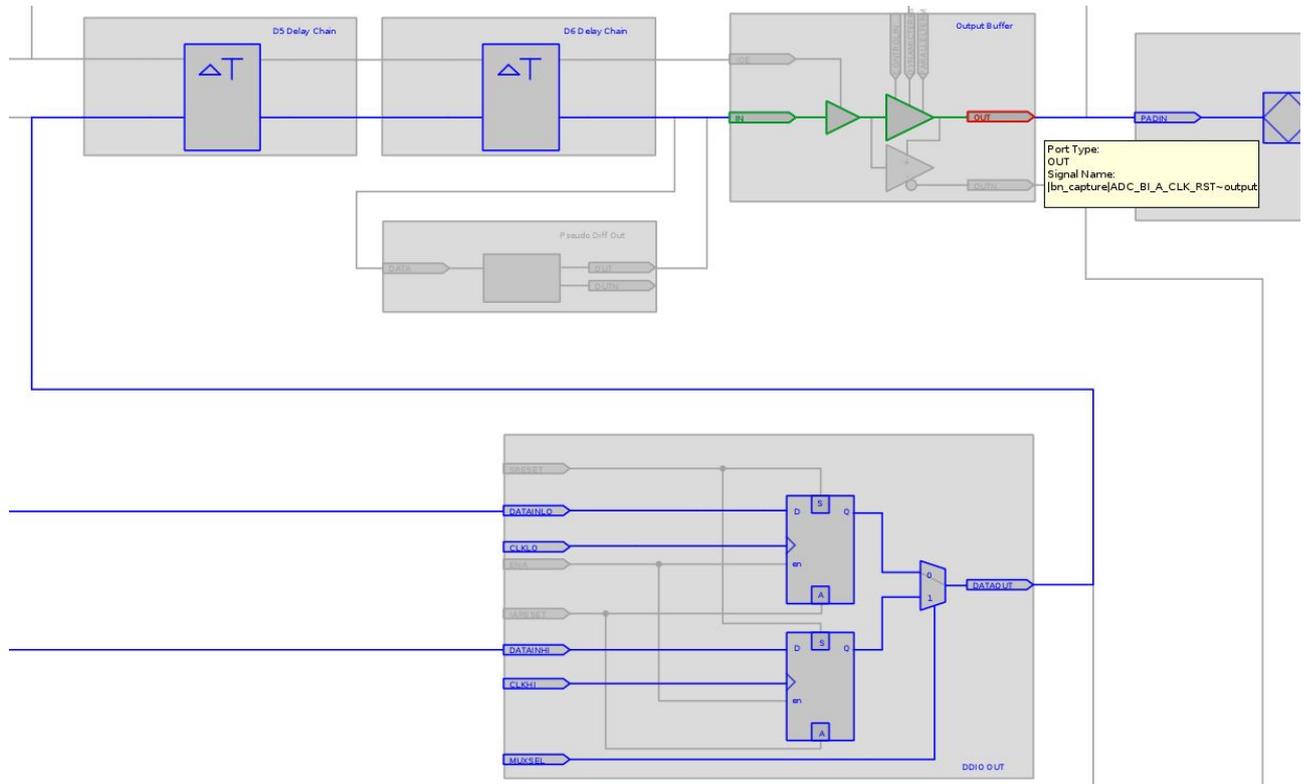


Figure 22: Output for ADC_BI_A_CLK_RST using D5 and DDIO

The timing between the 200 MHz *dp_clk* and the 400 MHz *dclk* domain is preserved thanks to the clock tree distribution network in the FPGA and by ensuring that the setup time for control signals of the mixed width FIFO is fixed. Therefore the FIFO reset signal is passed through a `common_acapture` component and the number of used words vector signal is passed through a `common_acapture_slv` component in Figure 32. Both the `common_acapture` and the `common_acapture_slv` fit inside 1 ALM, so by setting a logic-lock region constraint on these instances ensures that the data trace length between the signal in the *dclk* write clock domain and the *dp_clk* read clock domain is more or less fixed. The logic-lock regions are shown in Figure 23. Thanks to the clock tree network the location of the logic-lock region does not need to be constrained, only the size. The ALM is the smallest possible logic-lock region and it appears that the data trace within an ALM delay varies between 386 ps and 458 ps according to the TimeQuest Timing Analyzer, so about 72 ps which is acceptable compared to the *dclk* period of 2500 ps. Without the logic-lock region constraint the data trace length can span across the FPGA and result in several ns delay that then will also depend on the amount of logic in the rest of the design.

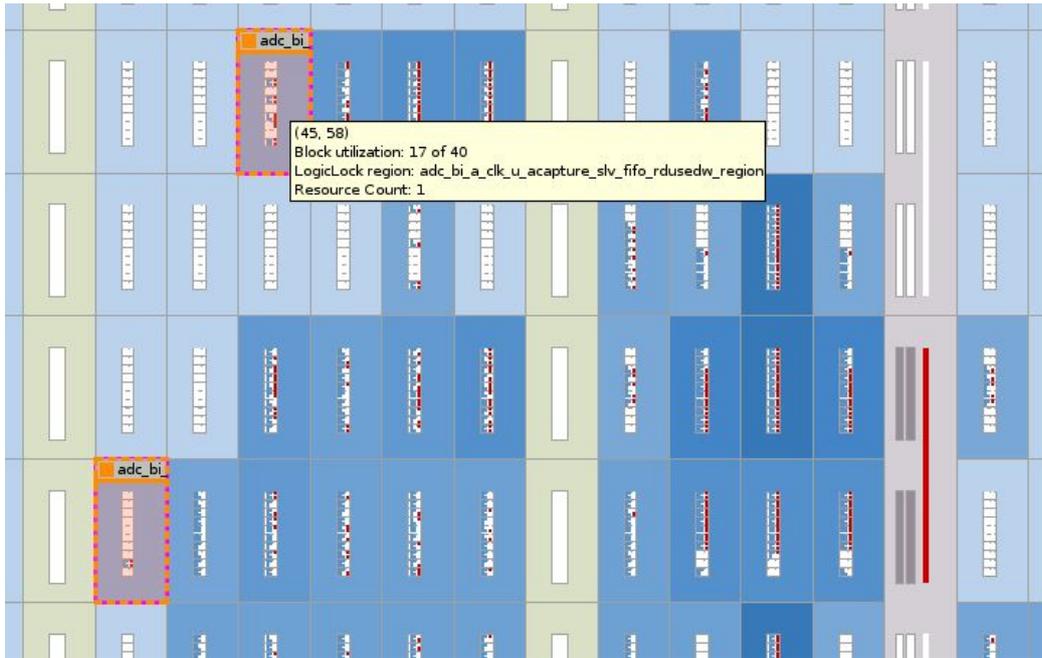


Figure 23: Lock regions using common_acapture

The logic-lock regions were set using LogicLock Region Window in the Quartus GUI and then saved into the TCL script:

```
$UNB/designs/bn_capture/src/tcl/ bn_capture_logic_lock_regions.tcl
```

That gets sourced by the node_bn_capture.qip file that included automatically in the bn_capture Quartus project, because all module QIP and node QIP files get included via:

```
$UNB/synth/quartus/unb_libraries.qip
```

2.4.3.6 Synthesis

Each PLL phase setting or IOE delay setting requires a separate FPGA image (i.e. a bn_capture.sof file). Synthesis of bn_capture without the DDR3 functionality takes about 15 minutes and can be done in a batch file using unb_qcomp). To be sure that indeed the file change due to a different setting is recognized by Quartus it is best to overwrite the bn_capture.qsf with the original project file with the original from SVN before starting a new synthesis.

2.4.4 Validation

The validation of the ADUH module with the UniBoard and ADU hardware is done with the `bn_capture` design [4].

2.4.4.1 Correct data

The sample data can be clocked in reliably, as demonstrated with the `bn_capture` design [4] on March 26, 2012. On 10 December 2012 the measurements were repeated using the `bn_capture` design on all 16 BN in the Apertif subrack and using the Python script `tc_bn_capture_adc_bist.py`:

```
> python apps/bn_capture/tc_bn_capture_adc_bist.py --unb 0:3 --bn 0:3 --sp 0:3 --rep 1 -n 300 -v 3
```

Figure 24 shows the BIST measurement results for all 64 SP and for 8 different settings of the input delay D3. One D3 unit ≈ 370 ps. Each BIST measurement was ran for at least 5 minutes and some for 1 hour. The dots in the table indicate that no sample bit errors occurred, so `verify_res[7:0] = 0` (see section 3.2). The hexadecimal values denote the `verify_res[7:0]` value when sample bit errors did occur. From the measurements it follows that D3 = 2 is a proper value for ADU-AB for all BN and D3 = 3 is a proper value for ADU-CD for all BN. The range of D3 settings shows the eye-diagram of the data, with the error free region and the transition region.

Figure 25 repeats the ADU \rightarrow BN data interface BIST measurements using the optimum timing settings from Table 2. The `clk` period is 1250 ps (800 MHz). The BIST measurements reveal that the eye-opening of stable data is about 800 ps wide and the transition region is about 450 ps wide. The Stratix IV FPGA LVDS interface is specified to operate at up to 1.6 Gbps. A sample period of $1/(1.6 \text{ GHz}) = 625$ ps with a transition region of 450 ps will still yield a (small) data eye-opening of 175 ps. Therefore based on the BIST measurements the ADUH can probably also work reliably for sample frequencies up to 1.6 GHz.

The BIST measurements were repeated several times for all 64 SP and no bit errors were ever detected. A test for 24 hours on 30/31 January 2013 also showed no bit errors. The log file of this test is printed in appendix 7.

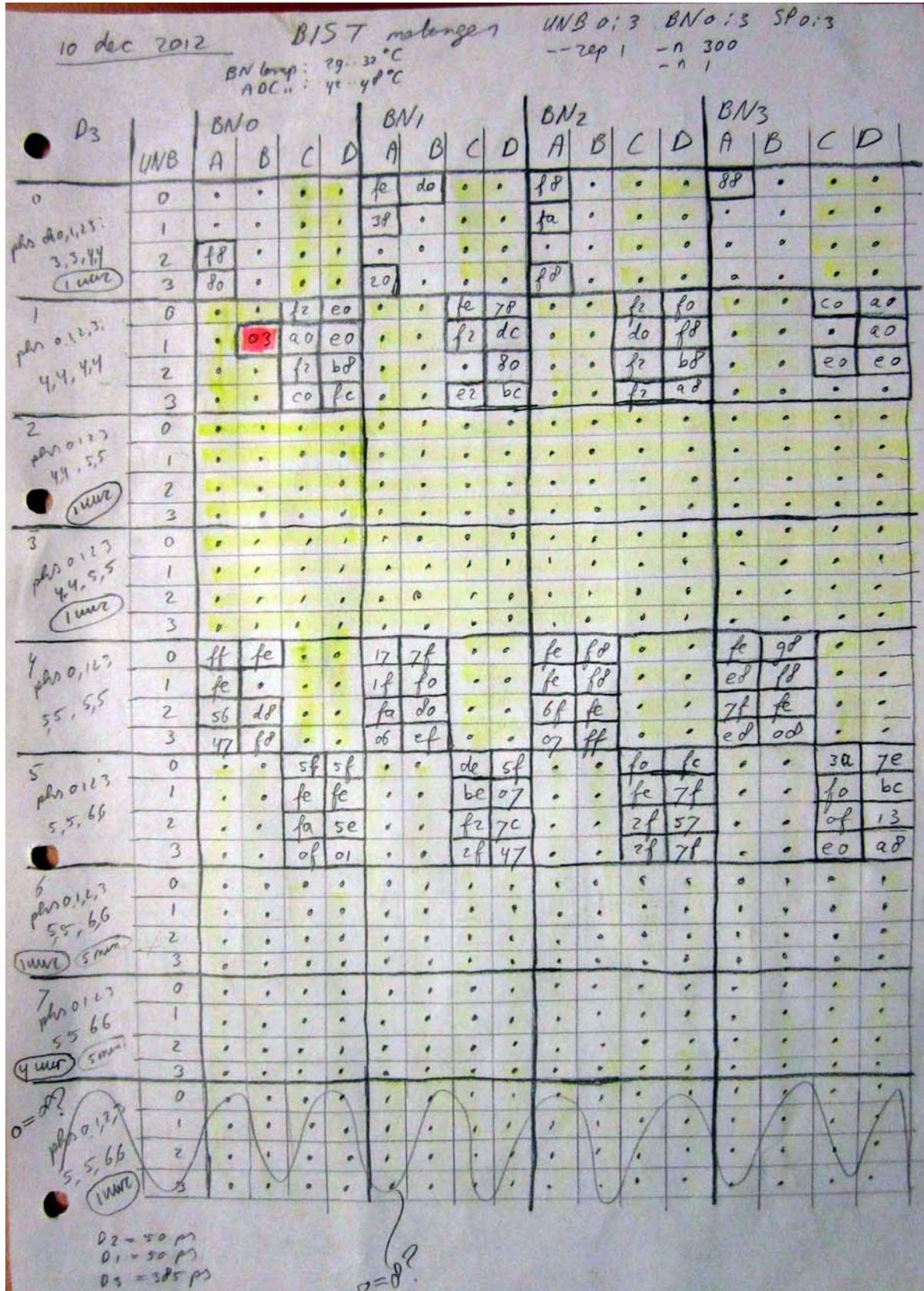


Figure 24: BIST measurement results for ADU → BN data interface using *dclk* and varying D3

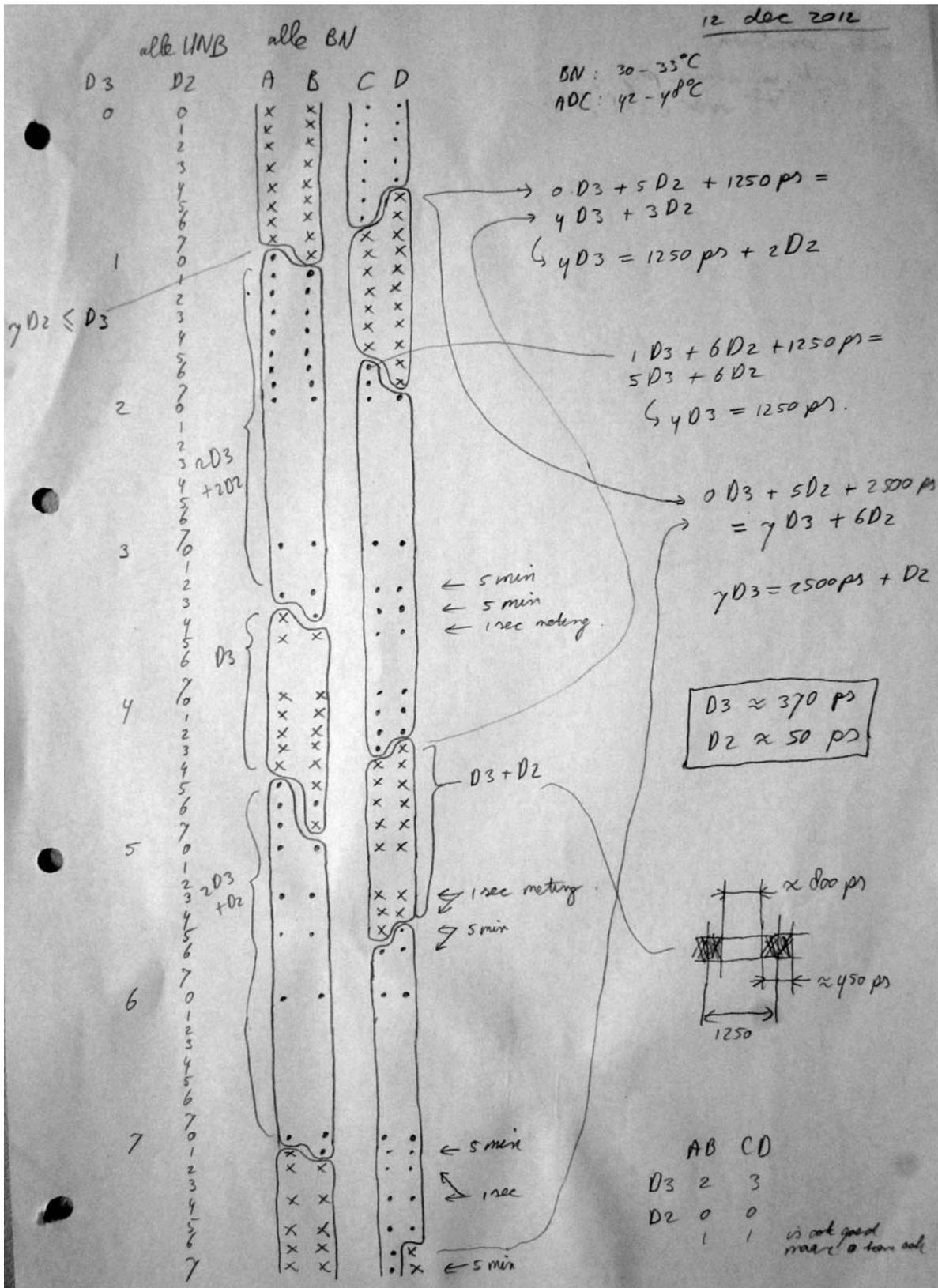


Figure 25: BIST measurement results for ADU → BN data interface using *dclk* and varying D3 and D2

2.4.4.2 Correct timing

The correct timing was measured for the set of SP marked in gray in Figure 3 by measuring the zero-crossings of the captured RF signal. The zero-crossings were measured for 32 different phase settings of the *dp_clk* PLL (see section 2.4.3.5) to cover the entire range of one *dp_clk* period, so 0 to 360 degrees in steps of 11.25 degrees (= 156 ps at 200 MHz). Figure 26 shows the nearest integer sample number at which the signal path zero-crossing occurred for 200 restarts of the ADCs using the `tc_bn_capture_adc_plot.py` Python script, e.g. like in:

```
> python apps/adu_tests/tc_bn_capture_adc_plot.py --unb 0 --bn 0 --sp 0,1,2,3 --rep 200 -v 3 -s try
```

The PLL phases for which the zero-crossing stayed at the same sample are grouped in Figure 26. The 'x' denote results for which the zero-crossing occurred at various sample phases. The '.' denote that at least once the zero-crossing occurred at another sample phase.

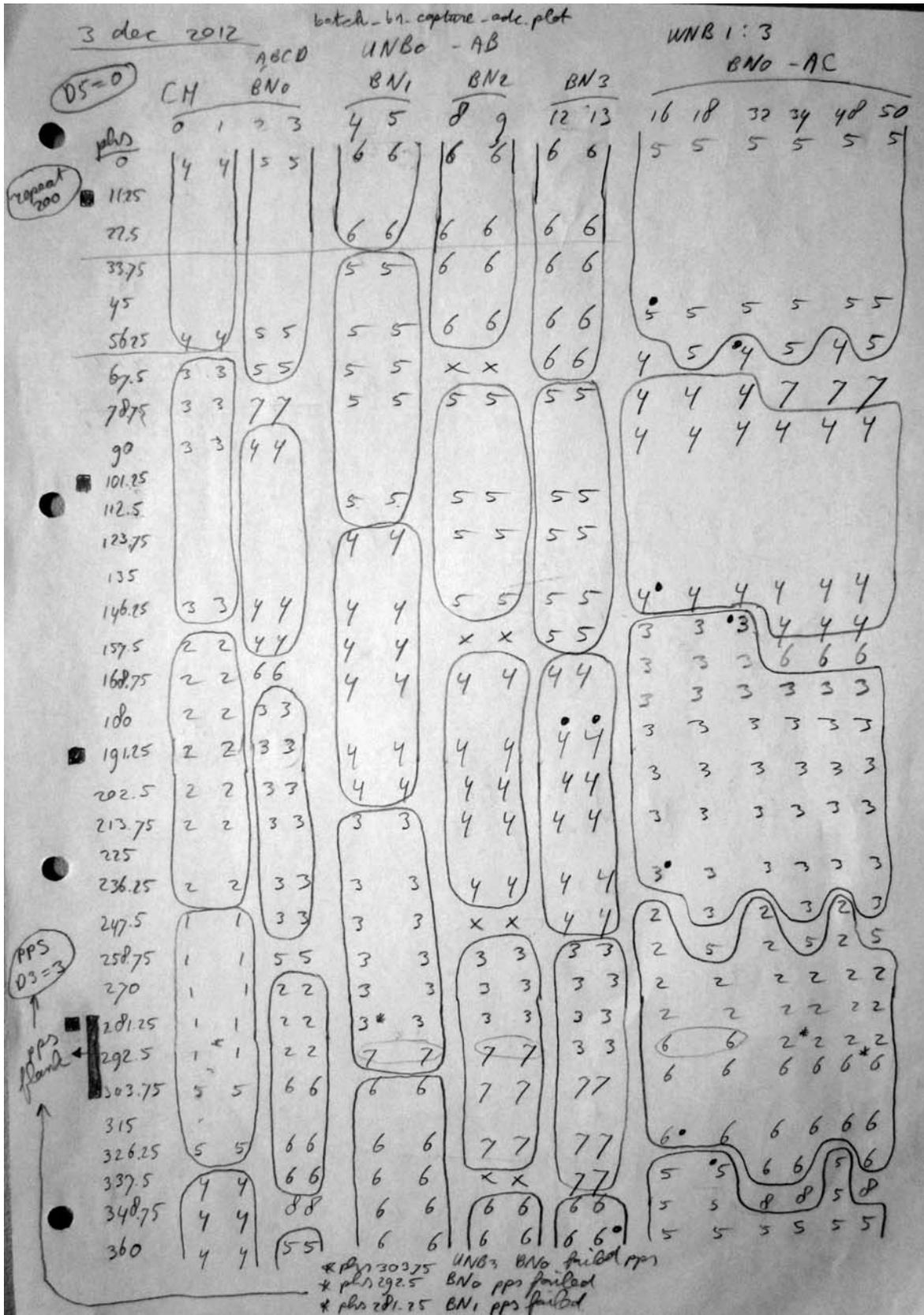


Figure 26: Sample phase measurements for different settings of the dp_clk PLL phase (D5=0)

Note that around PLL phase 292.5 degrees the sample phase gets a jump of 4 samples. This is due to that for that PLL setting the rising edge of external PPS lies close to the rising edge of the *dp_clk* that is used to clock in the PPS [6]. This PLL phase setting needs to be avoided for clocking in the PPS, or alternatively the falling edge of the *dp_clk* should be used to clock in the PPS. Clearly stable clocking in of the PPS has a large eye-diagram as shown in Figure 27, because only 3 out of the 32 PLL phases in Figure 26 coincide with the PPS rising edge. The PPS input delay setting for Figure 26 was D3=3, however the measurements reveal that it is fine to choose D3=0. With D3=0 there is even more margin with respect to the rising edge, because then the PPS transition will move to by about 1100 ps to the left in Figure 27 (one D3 unit \approx 370 ps) and is then expected to occur at about 180 degrees.

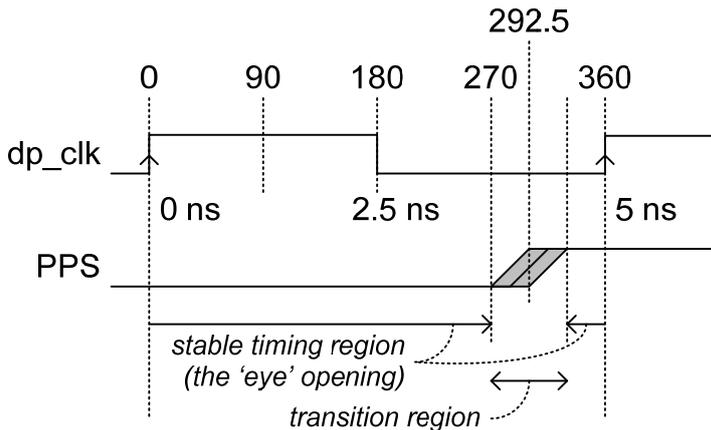


Figure 27: PPS eye-diagram for clocking with *dp_clk* and PPS input delay D3 = 3

From Figure 26 it follows that PLL phase setting of 11.25 degrees is suitable as common setting for all BN, because it yields a ± 11.25 degrees margin. Similar also 101.25 and 191.25 are suitable, because they lie 90 degrees or 1 sample period further. One would expect a much larger margin because all traces lengths throughout the subrack are matched (see section 2.4.1.2). However it appeared that the traces of the *dp_clk* and pps on the UniBoard are matched with respect to each other, but not between the BN0:3. Comparing in Figure 28 the actual trace delays on the UniBoard with the measurement results of Figure 26 clarifies why the margin across all BN is so small.

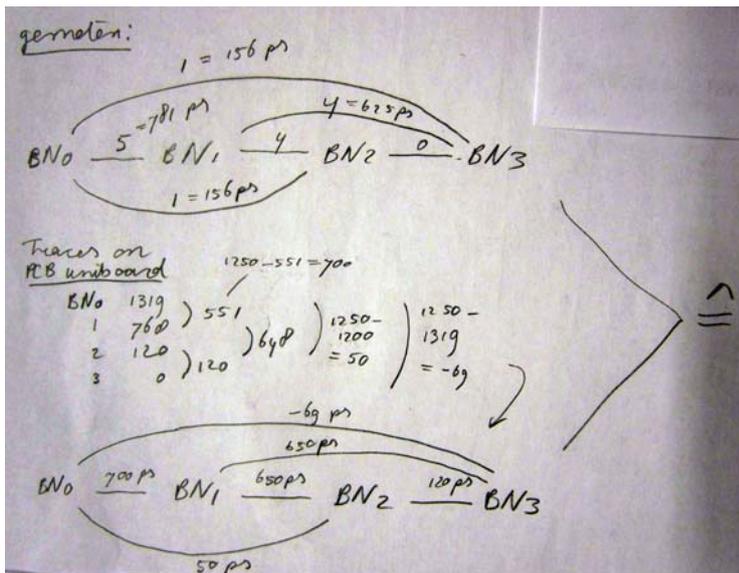


Figure 28: Correspondence in timing difference and trace length difference between BN3:0

For a next UniBoard the traces to the BN0:3 will be matched and then the margin can be expected to be as large as the smallest group of PLL phases with stable zero-crossings in Figure 26. The smallest group for BN2 spans from PLL phase 78.75 to 146.25, so about 67.5 degrees or about 0.9 ns. This eye-opening of about 0.9 ns is quite large compared to the sample clock period of 1250 ps, so also from a timing point of view it is expected that the ADUH can probably also work reliably for sample frequencies up to 1.6 GHz.

In Figure 26 the output delay setting for the *dclk_rst* signal is D5=0. In another set of measurements D5 was varied from 0 to 14 with *dp_clk* PLL phase fixed at 0 degrees. On BN1 D5=0:7 yielded sample phase 6, D5=9:14 yielded sample phase 5 and the transition occurred at D5=8 as shown in Figure 29. One D5 unit is about 45 ps and $8 \times 45 \text{ ps} = 360 \text{ ps}$ falls within the region that was also measured for BN1 in Figure 26. For *dp_clk* PLL phase 11.25 the output delay setting for *dclk_rst* D5 = 0 is suitable.

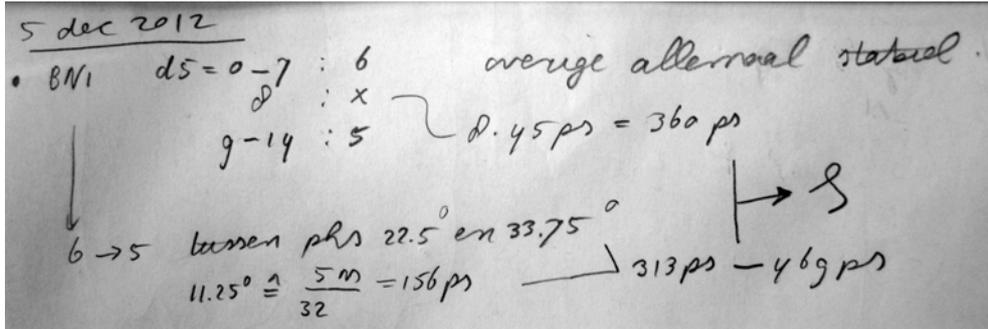


Figure 29: Sample phase measurements for PLL phase 0 and D5=0:14 for *dclk_rst*

For applications that instantiate node_bn_capture.vhd for using the ADUH with the Apertif subrack the optimum timing settings are listed in Table 2 (all other IOE delays are 0).

Signal	Setting
<i>dp_clk</i>	PLL phase = 11.25 degrees
<i>dclk_rst</i>	output delay D5 = 0
ADC_BI_A[7:0]	input delay D3 = 2
ADC_BI_B[7:0]	input delay D3 = 2
ADC_BI_C[7:0]	input delay D3 = 3
ADC_BI_D[7:0]	input delay D3 = 3
PPS	input delay D3 = 0

Table 2: Optimum settings for ADUH in Apertif subrack

2.5 Alternative solutions

2.5.1 Hardware aspects

Instead of using *dclk_rst* so control the phase of the *sclk* → *dclk* division one can accept the phase that occurs and then measure and compensate for it if it is not the desired phase. To measure the phase of the *dclk* with respect to the *sclk* requires clocking a central timing pulse from the *sclk* domain into the *dclk* domain. For this the PPS signal can be used. It is not necessary that the PAC board also distributes the PPS via the ADU boards to the BN, because the path delay is fixed (via traces and buffers) and the path delay value is not relevant as long as it is fixed. Therefore the PPS that already arrives from PAC at each UniBoard can also be used to reveal the phase of the *dclk* from ADU-0 and the *dclk* from ADU-1. It may be necessary to provide the PPS to the BN via up to three FPGA pins, because the DDR input logic in an IOE can only be used for external signals:

- one pin to clock in the PPS with the *dp_clk*
- one pin to clock in the PPS with the *dclk* from ADU-0
- one pin to clock in the PPS with the *dclk* from ADU-1

Instead of the PPS one may also choose to clock in the 200 MHz *clk* clock with the *dclk*, because that also provides the synchronization for the clock dividers. The two *dclk_rst* lines can then be removed from the ADU-BN interface.

In this scheme the system can be seen as having two synchronous samplers one for the RF data using the SDR *sclk* and one for the external time using the DDR *dclk*:

- the analogue time is represented by the PPS and is sampled by the 1-bit DDR input cell at both edges of the *dclk*
- the analogue RF signal is sampled by the 8-bit ADC at the rising edge of the *sclk*.

Once the PPS pulse is available in the *dclk* domain it is passed on along with the data through the mixed width FIFO into the *dp_clk* domain. In the *dp_clk* domain the samples can be shifted such that the PPS start falls on a 4-sample word boundary to complete the alignment of all SP between all ADU and all BN.

2.5.2 Firmware aspects

A risk with using fixed input and output delays is that these may vary too much with temperature. This could cause that it is impossible to reliably receive the ADC samples and to set the *dclk_rst* phase over the operational temperature range of typically 35 to 85 degrees. An alternative solution for the ADU Handler is then to use a PLL output to clock in the samples. Typically the input clock for the PLL is the *dclk* from ADU, but even the *dp_clk* could maybe be used, but then the lock detection is no longer available. Instead of delaying the data with an delay element in the IOE the capture clock phase is then adjusted by selecting the appropriated clock phase with a PLL. Typically this allows 8 or more clock phases for 1 period. There are three schemes with a PLL:

- non-DPA → select a fixed phase for the PLL output clock at synthesis
- DPA → Dynamic Phase Alignment, the optimal clock phase is adjusted dynamically based on signal transitions in the data
- Soft CDR → soft Clock Data Recover is issued when the clock is encoded within the data like e.g. for SGMII in a 1GbE receiver

Soft CDR is not applicable for ADU. DPA may be feasible, but seems less suitable because DPA requires that there are data transitions and with ADC data this is not guaranteed (e.g. in case of a DC signal). So non-DPA remains as PLL scheme.

On the ADU side of the BN FPGA there are 2 PLLs available, these are PLL_R3 for ADC_BI_A_CLK and PLL_R2 for ADC_BI_D_CLK. Problem is that PLL_R3 is already used for the TSE (Triple Speed Ethernet,

i.e. the 1GbE control interface) due to that the TSE data pins are located near PLL_R3. It is possible to use PLL_R2 also for the data from ADU-AB, but that implies that ADU-AB can then only be used if the ADC_BI_D_CLK clock of ADU-CD is active (because the ADC_BI_A_CLK is then not available). Two preliminary VHDL components aduh_pll and lvds_pll are available, but these need further development. It was considered for a next release of the UniBoard hardware to move the TSE data pins to another PLL, to free up PLL_R3 for the ADC_BI_A_CLK. However it was decided that compared to the benefit this takes too much effort regarding PCB layout and simulation and synthesis to check that it fits in the FPGA.

3 MMS_ADUH_QUAD

The ADU handler (ADUH) for the 4 ADC input signal paths is called `aduh_quad`. The `mms_aduh_quad` provides the MM register interface to the `aduh_quad` and is shown in Figure 30.

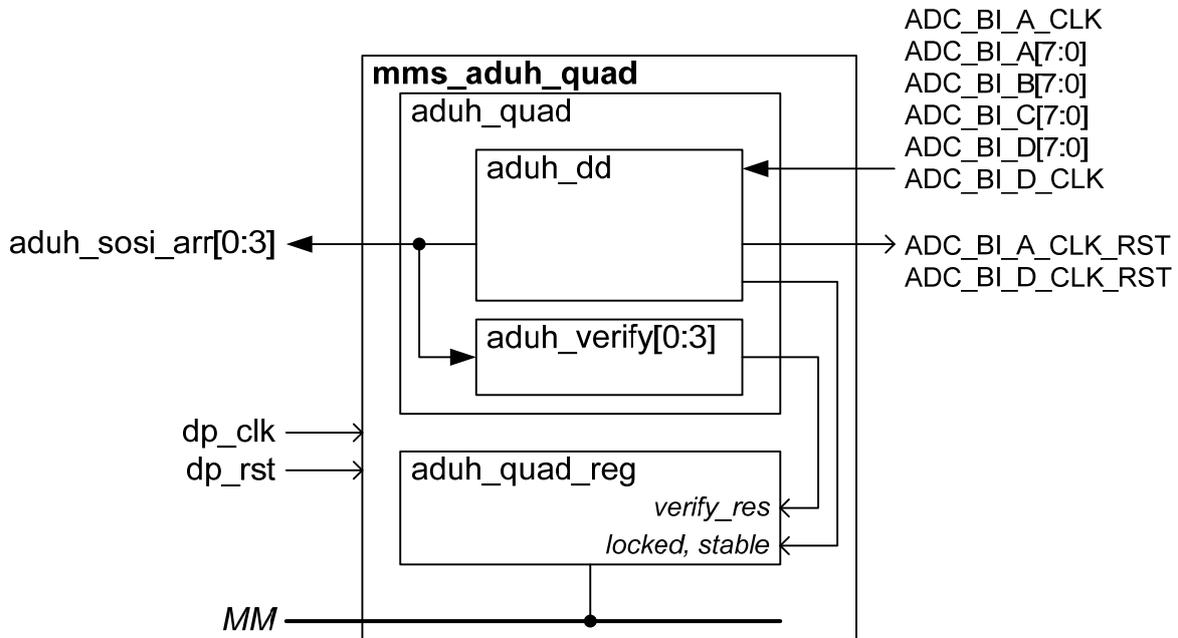


Figure 30: Block diagram of `mms_aduh_quad`

3.1 ADUH_QUAD

The `aduh_quad` component instantiates one `aduh_dd` component for receiving the ADC samples from two signal paths from ADU-AB and from two signals paths from ADU-CD. The `aduh_dd` is described in section 4. For each of the 4 signal paths the `aduh_quad` instantiates an `aduh_verify` component that can automatically verify the test pattern that the `adc08d1020` ADC that is used on ADU can output to validate the ADU-ADC to UniBoard-BN LVDS interface. The `aduh_verify` is described in section 5.

3.2 ADUH_QUAD_REG

The `aduh_quad` can be accessed via the MM bus. The register map is defined by `aduh_quad_reg`. In a BN FPGA design the registers can be accessed with Python using the `pi_aduh_quad.py` peripheral. The register map supports:

- Report the locked status for ADU-AB and ADU-CD
- Report pattern verification result for each ADC [A,B,C,D]

31	24 23	16 15	8 7	0	wi
xxx			ab_stable & ab_locked = [1:0]		0
xxx			cd_stable & cd_locked = [1:0]		1
xxx	a_verify_res_val[12]	xxx	a_verify_res[8:0]		2
xxx	b_verify_res_val[12]	xxx	b_verify_res[8:0]		3
xxx	c_verify_res_val[12]	xxx	c_verify_res[8:0]		4
xxx	d_verify_res_val[12]	xxx	d_verify_res[8:0]		5

Table 3: Memory map for REG_ADC_QUAD

The ADUH sample receiver monitors the activity of the *dclk* from ADU. When the *dclk* is in phase with the *dp_clk* then *locked* = '1'. If *locked* = '1' for every clock cycle since the last read access to *wi* = 0 for ADU[0] or *wi* = 1 for ADU[1] then *stable* = '1'. Else if *stable* = '0' then at least once since the last read access the *dclk* and *dp_clk* lost lock. If *locked* = '0' then the *dclk* is not in phase with the *dp_clk* or not connected.

The ADUH verify function implements the build-in self-test (BIST) that continuously verifies the received samples from the ADU ADCs and reports whether they match the ADU ADC test pattern. If a mismatch occurs then the result remains indicating this error. A new verification interval starts when the result register has been read. The *verify_res_val* = '1' when there was an active *dclk* from the ADU. The *verify_res*[7:0] bits are '0' when the corresponding sample bit matches the test pattern and becomes '1' when the bit mismatches. The *verify_res*[8] bit reports the result for the entire 8 bit sample (so it is equivalent to the vector-or operation of *verify_res*[7:0]). If *verify_res_val* = '0' then the *verify_res* must be ignored.

The Software/python/peripherals/pi_aduh_quad.py Python peripheral script provides methods for accessing the ADUH_QUAD registers and the util_aduh_quad.py provides usage examples.

3.3 Verification

The test bench `tb_mms_aduh_quad` verifies the `mms_aduh_quad` by using the `adc08d1020` test pattern mode. The test bench uses a behavioral model in VHDL of the National `adc08d1020` ADC on the ADU (`adc08d1020.vhd` and `adu_half.vhd`) and an `aduh_quad_scope` in VHDL to more easily view the 800 MHz ADC signal as an analogue signal in the Modelsim Wave window.

The `$UNB/Firmware/software/apps/bn_capture/main.c` program also verifies the ADC test pattern for the samples that are captured in the ADU monitor buffer using the `aduh_verify_adc_test_pattern()` function in the `aduh.c` module. This software function is now no longer used, because the `aduh_verify` VHDL component can now do the verification continuously in real time for all samples at the sample rate.

4 ADUH_DD - ADC data receiver

4.1 ADUH_DD

The ADUH_DD use DDIO input (no PLL) to receive the samples from the LVDS Rx interface between a BN and two ADCs on one ADU.

4.1.1 Hardware interface

The aduh_dd interface parameters and ports are given in respectively Table 4 and Table 5.

Generic field	Type	Description
nof_sp	natural	Fixed support 4 signal paths A,B,C,D, whether they contain active data depends on nof_adu.
nof_adu	natural	When 2 ADUs then use all 4 ports A,B,C,D, one ADU on ports A,B and one ADU on ports C,D, when 1 ADU then only use ports C,D.
nof_ports	natural	Fixed 2 ADC BI ports per ADU.
port_w	natural	Fixed 8 bit ADC BI port width, the ADC sample width is also 8 bit.
dd_factor	natural	Fixed 2. Fixed double data rate factor for lvds data (800 MSps) and lvds clock (400 MHz).
rx_factor	natural	Default 2. When 1 then the data path processing clock frequency is 400 MHz (= lvds clock / 1), when 2 then the data path processing clock frequency is 200 MHz (= lvds clock / 2).
clk_rst_enable	natural	Default true for initial DCLK_RST pulse to control the ADC DCLK phase, else false for no DCLK_RST pulse.
clk_rst_invert	natural	Default false because DCLK_RST pulse on ADU is active high, use true for active low pulse to compensate for P/N cross
deskew	t_c_aduh_delays	Input de-skew buffer delays. Default all 0 because the input delays are set via IOE constraints during synthesis.

Table 4: aduh_dd parameter g_ai fields of the record type t_c_aduh_dd_ai

Signal	IO	Type	Description
ADC_BI_A[7:0]	IN	std_logic_vector	Input sample from ADC-I on ADU-AB
ADC_BI_B[7:0]	IN	std_logic_vector	Input sample from ADC-Q on ADU-AB
ADC_BI_C[7:0]	IN	std_logic_vector	Input sample from ADC-I on ADU-CD
ADC_BI_D[7:0]	IN	std_logic_vector	Input sample from ADC-Q on ADU-CD
ADC_BI_A_CLK	IN	std_logic	Input 400 MHz DDR lvds clock from the ADC on ADU-AB
ADC_BI_D_CLK	IN	std_logic	Input 400 MHz DDR lvds clock from the ADC on ADU-CD
ADC_BI_A_CLK_RST	OUT	std_logic	Reset pulse to synchronises ADU_AB lvds clock divider
ADC_BI_D_CLK_RST	OUT	std_logic	Reset pulse to synchronises ADU_CD lvds clock divider
ab_locked cd_locked	OUT	std_logic	When active then the <i>dp_clk</i> in the BN and the lvds clock from the ADU are currently synchronous and the data samples are being received. Idem for ADU-CD.
ab_stable cd_stable	OUT	std_logic	When active then the <i>ab_locked</i> has been continuously active since the last time that <i>ab_stable_ack</i> was pulsed, else when inactive then <i>ab_locked</i> has gone active at least once. Idem for ADU-CD.
ab_stable_ack cd_stable_ack	IN	std_logic	A pulse restarts a new period of monitoring <i>ab_locked</i> via <i>ab_stable</i> . Idem for ADU-CD.
dp_rst	IN	std_logic	Data path reset
dp_clk	IN	std_logic	Data path clock (200 MHz)
src_out_arr[0:3]	OUT	t_dp_sosi_arr	Output sample for signal inputs [A,B,C,D] = [0:3]. The data field contains 4 time series 8 bit ADC samples packed in big-endian format as [t0&t1&t2&3] = [31:0]. The valid field indicates whether the data is valid. The other sosi fields are not used.

Table 5: aduh_dd IO

4.1.2 Design

The aduh_dd receives ADC samples for 4 signals paths as shown in Figure 31. Each BN on UniBoard processes 4 signals paths but it gets these from two ADUs, therefore the aduh_dd instantiates two lvds_dd components, one for each ADU.

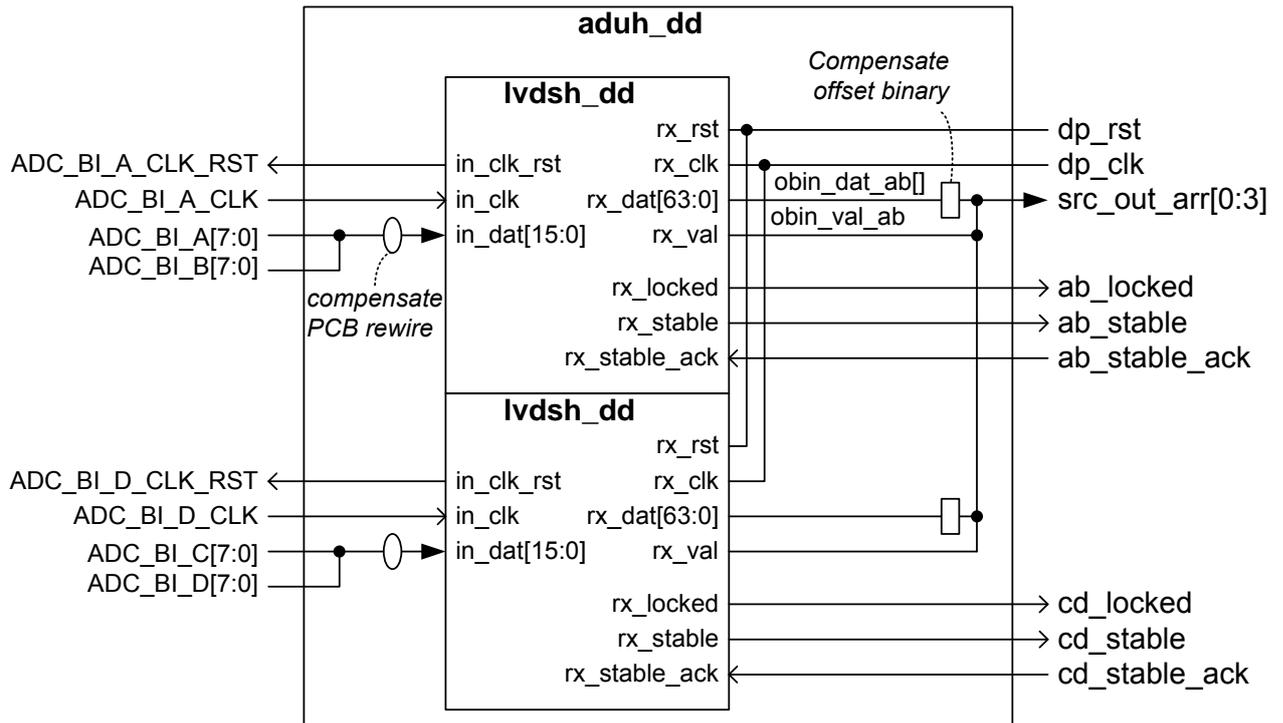


Figure 31: Block diagram of aduh_dd

The aduh_dd takes care of:

- Receiving the ADC samples for signal paths A, B, C and D using the lvdsh_dd
- Compensating of PCB rewiring of the ADC data of signal paths B and D on ADU
- Converting the offset binary format of the ADC data into two-complement data by inverting the MSbit
- Default it outputs the CLK_RST pulse to the ADC as active high

4.1.3 Implementation

The ADUH_DD uses Double Data rate IO (so no PLL) to capture the LVDS input data. Using the DDR in the IOE and static IO delay settings was chosen because it is feasible to directly use the data clock and because it feasible to use static IO delay settings. The ADUH_DD also provides:

- Input locked status indicates whether the input clock is currently in phase with the processing clock
- Input locked stable indicates whether the input locked status has been locked since the last time that the host read the input locked status
- Apply *dclk_rst* pulse when input clock activity has been detected.

The input locked functionality and *dclk_rst* functionality depend on the input FIFO fill level. If an ADU is removed and reinserted then the input will lock automatically and the *dclk_rst* pulse will also be applied automatically, i.e. no involvement by the control computer is needed for this. The control computer can know that something happened by monitoring the input locked stable status bit.

4.2 LVDSH_DD

4.2.1 Design

Figure 32 shows a block diagram of the `lvdsh_dd`. The `common_acapture` instances get a logic-lock region constraint to ensure that the signal is transferred with fixed time delay between the $dclk = in_clk$ domain and the $dp_clk = rx_clk$ domain as explained in section 2.4.3.5.

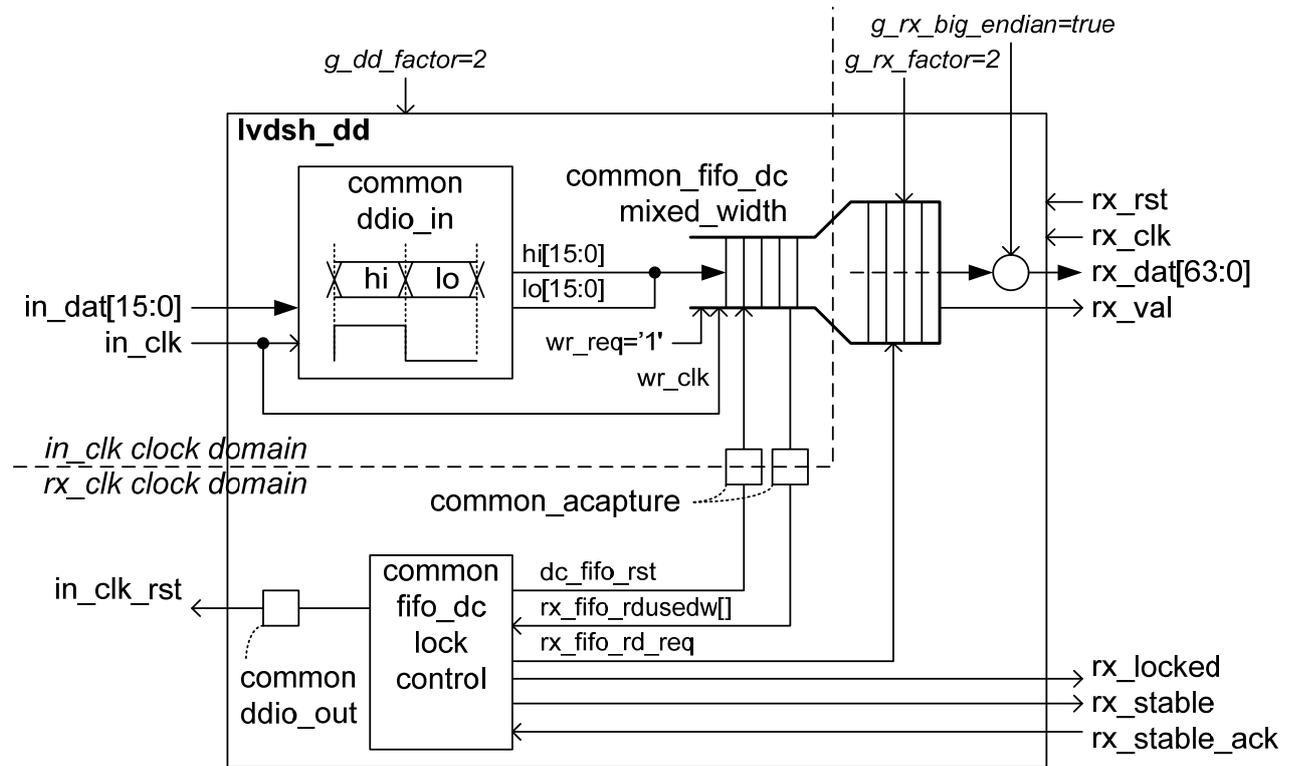


Figure 32: Block diagram of `lvdsh_dd`

4.2.2 Verification

The test bench `tb_lvdsd_dd.vhd` verifies the `lvdsh_dd`.

4.3 ADUH_PLL

Preliminary component for the alternative solution using a PLL, see section 2.5.

4.4 LVDSH_PLL

Preliminary component for the alternative solution using a PLL, see section 2.5.

5 ADUH_VERIFY - ADC test pattern verification

5.1 Hardware interface

The `aduh_verify` interface parameters and ports are given in respectively Table 6 and Table 7.

Generic	Type	Description
<code>g_symbol_w</code>	natural	Fixed 8 bit. The ADC sample width.
<code>g_nof_symbols_per_data</code>	natural	Fixed, big endian <code>in_sosi.data</code> , <code>t0</code> in <code>MSSymbol</code> , so <code>[h:0] = [31:0] = [t0]&[t1]&[t2]&[t3]</code>
<code>deskew</code>	<code>t_c_aduh_delays</code>	Input de-skew buffer delays. Default all 0 because the input delays are set via IOE constraints during synthesis.

Table 6: `aduh_verify` parameters

Signal	IO	Type	Description
<code>rst</code>	IN	<code>std_logic</code>	Data path reset
<code>clk</code>	IN	<code>std_logic</code>	Data path clock 200 MHz
<code>in_sosi</code>	IN	<code>t_dp_sosi</code>	Signal path data with 4 800MHz 8b samples in time per one 32b word @ 200MHz
<code>pattern_sel</code>	IN	natural	Selects ADC port from which the data comes, 0 = DI, 1 = DQ
<code>verify_res[8:0]</code>	OUT	<code>std_logic_vector</code>	The <code>verify_res[8]</code> contains the verify result for the aggregate symbol values, and <code>verify_res[7:0]</code> contains the result per corresponding symbol bit [7:0].
<code>verify_res_val</code>	OUT	<code>std_logic</code>	When high then <code>verify_res</code> is valid else it is undefined.
<code>verify_res_ack</code>	IN	<code>std_logic</code>	The duration of the test pattern verification interval depends on <code>verify_res_ack</code> , each time <code>verify_res_ack</code> pulses a new verification interval starts.

Table 7: `aduh_verify` IO

5.2 Design

The `aduh_verify` uses multiple instances of the `aduh_verify_bit` to verify the `adc08d1020` test pattern “0 1 0 0 1 1 0 0 1 0” per LVDS data bit. The `adc08d1020` on ADU has two ADCs I and Q, so it outputs two signal paths. Both ADC I and Q use different test patterns, therefore via `pattern_sel` one can select whether to verify for the I signal path or the Q signal path. The test pattern data is periodic over 10 samples:

	TP_I	TP_Q	TP_OV
T0	02h	01h	0
T1	FDh	FEh	1
T2	02h	01h	0
T3	02h	01h	0
T4	FDh	FEh	1
T5	FDh	FEh	1
T6	02h	01h	0
T7	02h	01h	0
T8	FDh	FEh	1
T9	02h	01h	0

The verification is always ready to accept data, therefore it has no `in_siso` output. The verification is always enabled. After reset and when `verify_res_ack` pulses then `verify_res_val = '0'`. The verification needs two words to initialize its local reference pattern generator and then the next words can be verified. At the third

valid input word the *verify_res_val* goes active '1' and remains active until the next *verify_res_ack* pulse. If the received data is a mismatch with the local reference pattern then the *verify_res* goes high '1' and remains '1' until the next *verify_res_ack* pulse. The *verify_res[8]* contains the matching result for the aggregate symbol values, and *verify_res[7:0]* contains the result per corresponding symbol bit [7:0]. Via *verify_res[7:0]* the skew between LVDS input lines can be measured. Via *verify_res[8]* it becomes clear whether the skew is sufficiently small to have an open sampling eye for the entire symbol value. The duration of the verification interval depends on *verify_res_ack*, each time *verify_res_ack* pulses a new verification interval starts. The ADC overflow bit is not verified.

5.3 Implementation

The TP_I and TP_Q test symbols effectively only contain two values (0x02, 0xFD) or (0x01, 0xFE) respectively. Hence these can be mapped on single bit values '0' and '1' as is done via the signal *ymb*. The TP_I and TP_Q symbols are verified per bit and for the entire symbol via the mapped *ymb* signal. The 8 symbol bits and the mapped *ymb* signal all have the test pattern of 10 values: "0 1 0 0 1 1 0 0 1 0". The data arrives with *g_nof_symbols_per_data*=4 symbols per data, so a sequence of two test patterns (2*10 divides by 4) can appear at 10 different phases as:

Phase	Pattern	Pattern	Nibble hex values
0	0100 1100	1001 0011 0010	= 4 C 9 3 2
1	1001 1001	0010 0110 0100	= 9 9 2 6 4
2	0011 0010	0100 1100 1001	= 3 2 4 C 9
3	0110 0100	1001 1001 0010	= 6 4 9 9 2
4	1100 1001	0011 0010 0100	= C 9 3 2 4
5	1001 0010	0110 0100 1001	= 9 2 6 4 9
6	0010 0100	1100 1001 0011	= 2 4 C 9 3
7	0100 1001	1001 0010 0110	= 4 9 9 2 6
8	1001 0011	0010 0100 1100	= 9 3 2 4 C
9	0010 0110	0100 1001 1001	= 2 6 4 9 9

Hence for phase 0 to 9 the 4-bit nibbles can either be repeated <4 C 9 3 2> or <9 9 2 6 4>. E.g. if the first two data words map to 4 C then the next expected data word is 9. One data word (i.e. 4 symbols of the 10) is not enough to know the next test pattern data word. Two data words (i.e. 8 symbols of the 10) are sufficient to know the next test pattern data word. This is implemented by *func_tp_seq* in *aduh_verify_bit*.

5.4 Verification

The test bench *tb_aduh_verify.vhd* verifies the *aduh_verify*.

5.5 Validation

The *adc08d1020* can be programmed into test pattern mode with Python using the *pi_adu_i2c_commander.py* peripheral and an *i2c_commander* VHDL instance in the FPGA design (as in *bn_capture* [4]). The *pi_aduh_quad.py* peripheral then provides the methods to check *verify_res* as done with *util_aduh_quad.py*.

6 MMS_ADUH_MONITOR

6.1 ADUH_MONITOR

For one input signal path the `aduh_monitor` provides provides MM access to:

- Mean sum
- Power sum
- Data buffer time samples stored in big endian order:

The new *mean_sum* and the *power_sum* stored at a sync pulse. Hence the sync pulse interval determines the integration time. The data buffer is filled at a sync pulse.

The mean sum is calculated by the `aduh_mean_sum` component and the power sum is calculated by the `aduh_power_sum` component. Both components are capable of handling *g_nof_symbols_per_data* ≥ 1 , e.g. 4 samples per data word.

6.2 ADUH_MONITOR_REG

The register map for the `aduh_monitor` is defined by the `aduh_monitor_reg` VHDL component and can be accessed in Python using `pi_aduh_monitor.py`.

Table 8 shows register map for reading the mean and power statistics of the signal samples. The mean sum accumulates the ADC sample values during a sync interval. The power sum accumulates the squares of the ADC sample values during a sync interval.

31	24	23	16	15	8	7	0	wi
----- ----- ----- -----								
mean_sum[31:0]								0
----- ----- ----- -----								
mean_sum[63:32]								1
----- ----- ----- -----								
power_sum[31:0]								2
----- ----- ----- -----								
power_sum[63:32]								3
----- ----- ----- -----								

Table 8: Memory map REG_ADUH_MON for mean and power statistics

Table 9 shows the register map for the samples monitor buffer. The samples monitor buffer is a small internal FPGA buffer that can store 1024 samples per signal path and gets triggered to do so by the sync pulse at the start of every sync interval. The time samples are stored in big endian order, so when the 32-bit word is read in hexadecimal format it shows the sample values in time from left to right.

31	24	23	16	15	8	7	0	wi
t0[7:0]		t1[7:0]		t2[7:0]		t3[7:0]		0
t4[7:0]		t5[7:0]		t6[7:0]		t7[7:0]		1
...								..
t1020[7:0]		t1021[7:0]		t1022[7:0]		t1023[7:0]		255

Table 9: Memory map RAM_ADUH_MON for data buffer monitor

The Software/python/peripherals/pi_aduh_monitor.py Python peripheral script provides methods for accessing the ADUH_MONITOR registers and the util_aduh_monitor.py provides usage examples.

7 Appendix : Data value measurement results using the BIST

This appendix shows the log file of a successful BIST measurement with test pattern data from ADUs on all 64 signal paths in the Apertif subrack. Note that it is better to repeat a test 24 times for 1 hour than to run it once for 24 hours. By measuring for shorter intervals one gets an impression of the bit error distribution in time in case they occur.

```
> python apps/bn_capture/tc_bn_capture_adc_bist.py --unb 0:3 --bn 0:3 --sp 0:3 --rep 24 -n 3600 -v 3
```

```
[2013:01:30 12:27:01] - (3) TC_BN_CAPTURE -
```

```
>>> Title : Test case to run the ADC-[ABCD] BIST on UNB-[0, 1, 2, 3], BN-[0, 1, 2, 3], SP-[0, 1, 2, 3]:
```

```
SENS - UNB-0, BN-0: FPGA temperature = 31 [degrees]
SENS - UNB-0, BN-1: FPGA temperature = 33 [degrees]
SENS - UNB-0, BN-2: FPGA temperature = 32 [degrees]
SENS - UNB-0, BN-3: FPGA temperature = 31 [degrees]
SENS - UNB-1, BN-0: FPGA temperature = 31 [degrees]
SENS - UNB-1, BN-1: FPGA temperature = 32 [degrees]
SENS - UNB-1, BN-2: FPGA temperature = 31 [degrees]
SENS - UNB-1, BN-3: FPGA temperature = 31 [degrees]
SENS - UNB-2, BN-0: FPGA temperature = 32 [degrees]
SENS - UNB-2, BN-1: FPGA temperature = 33 [degrees]
SENS - UNB-2, BN-2: FPGA temperature = 29 [degrees]
SENS - UNB-2, BN-3: FPGA temperature = 0 [degrees]
SENS - UNB-3, BN-0: FPGA temperature = 31 [degrees]
SENS - UNB-3, BN-1: FPGA temperature = 32 [degrees]
SENS - UNB-3, BN-2: FPGA temperature = 30 [degrees]
SENS - UNB-3, BN-3: FPGA temperature = 31 [degrees]
ADUH_QUAD - UNB-0, BN-0: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-0, BN-0: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-0, BN-1: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-0, BN-1: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-0, BN-2: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-0, BN-2: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-0, BN-3: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-0, BN-3: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-1, BN-0: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-1, BN-0: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-1, BN-1: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-1, BN-1: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-1, BN-2: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-1, BN-2: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-1, BN-3: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-1, BN-3: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-2, BN-0: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-2, BN-0: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-2, BN-1: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-2, BN-1: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-2, BN-2: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-2, BN-2: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-2, BN-3: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-2, BN-3: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-3, BN-0: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-3, BN-0: ADUH-CD is locked and stable (3)
```

ADUH_QUAD - UNB-3, BN-1: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-3, BN-1: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-3, BN-2: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-3, BN-2: ADUH-CD is locked and stable (3)
ADUH_QUAD - UNB-3, BN-3: ADUH-AB is locked and stable (3)
ADUH_QUAD - UNB-3, BN-3: ADUH-CD is locked and stable (3)
Rep = 0
Rep = 1
Rep = 2
Rep = 3
Rep = 4
Rep = 5
Rep = 6
Rep = 7
Rep = 8
Rep = 9
Rep = 10
Rep = 11
Rep = 12
Rep = 13
Rep = 14
Rep = 15
Rep = 16
Rep = 17
Rep = 18
Rep = 19
Rep = 20
Rep = 21
Rep = 22
Rep = 23

>>> Test case result: PASSED (run time 86405 s)

8 Appendix : Data timing measurement results using a CW

8.1 SP 0, 1, 2, 3

Results of Figure 6, Figure 9, Figure 12, Figure 15 and Figure 18.

[2012:12:11 21:57:42] - (3) TC_BN_CAPTURE_AND_PLOT -

>>> Title : Test case to plot the ADC-[ABCD] data for UNB-[0], BN-[0], SP-[0, 1, 2, 3]:

Rep-0

Rep-600

Rep-1200

Rep-1800

Rep-2400

Rep-2999

SENS - UNB-0, BN-0: FPGA temperature = 36 [degrees]

ADU_I2C - UNB-0, BN-0, ADU-AB: Read temperature = 42 [degrees]

ADU_I2C - UNB-0, BN-0, ADU-CD: Read temperature = 45 [degrees]

PPS - UNB-0, BN-0: read_ppsh_stable = OK

>>> Clock-CW DCs:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	-0.42816	0.07029	-0.14648	-0.67676	0.53027	912	153	10	1	
CH1	-0.60260	0.08457	-0.32031	-0.91211	0.59180	974	141	7		
CH2	-0.04799	0.08183	0.28809	-0.38770	0.67578	926	153	16	2	
CH3	-0.86403	0.07613	-0.59277	-1.11719	0.52441	973	124	7		

>>> Clock-CW phases:

Channel	Mean	Std	Max	Min	Diff	Nof>1U	Nof>2U	Nof>3U	Nof>4U	Nof>5U
CH0	4.32410	0.00388	4.34020	4.30777	0.03244					
CH1	4.38830	0.00445	4.40386	4.37333	0.03053					
CH2	4.86471	0.00417	4.87885	4.84813	0.03071					
CH3	4.96129	0.00376	4.97684	4.94975	0.02708					

>>> Clock-CW amplitudes:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	50.31613	0.09430	50.74535	49.94976	0.79559	895	140	18	2	
CH1	49.53958	0.11676	49.86494	49.17608	0.68886	985	125	4		
CH2	48.86487	0.19931	49.25119	48.25059	1.00060	1177	66	1		
CH3	48.25007	0.17036	48.74476	47.83433	0.91043	1141	50			

>>> Clock-CW noise peaks:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	1.17836	0.09860	1.70369	0.68915	1.01454	741	150	40	11	3
CH1	1.14815	0.18439	1.66091	0.68075	0.98016	1017	103			
CH2	1.12295	0.13106	1.88777	0.66579	1.22198	905	129	27	3	1
CH3	1.14555	0.14022	1.64152	0.72222	0.91929	858	178	13		

>>> Clock-CW SNRs:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	48.27535	1.03312	51.62446	44.23722	7.38724	932	134	11		
CH1	46.69371	0.59305	49.17417	44.91437	4.25980	918	143	16	1	
CH2	46.84892	1.37192	52.79533	43.98593	8.80941	901	124	26	2	
CH3	47.21466	0.83819	50.55478	44.13033	6.42445	914	160	21		

>>> Test case result: PASSED (run time 32232 s)

8.2 SP 0, 1, 4, 5, 8, 9, 12, 13

Results of Figure 7, Figure 10, Figure 13, Figure 16 and Figure 19.

[2012:12:12 06:54:55] - (3) TC_BN_CAPTURE_AND_PLOT -

>>> Title : Test case to plot the ADC-[ABCD] data for UNB-[0], BN-[0, 1, 2, 3], SP-[0, 1]:

Rep-0

Rep-600

Rep-1200

Rep-1800

Rep-2400

Rep-2999

SENS - UNB-0, BN-0: FPGA temperature = 36 [degrees]

SENS - UNB-0, BN-1: FPGA temperature = 37 [degrees]

SENS - UNB-0, BN-2: FPGA temperature = 36 [degrees]

SENS - UNB-0, BN-3: FPGA temperature = 35 [degrees]

SENS - UNB-0, BN-3: ETH PHY temperature = 45 [degrees]

SENS - UNB-0, BN-3: UNB supply current = 2.3 [A]

SENS - UNB-0, BN-3: UNB supply voltage = 48.0 [V]

ADU_I2C - UNB-0, BN-0, ADU-AB: Read temperature = 42 [degrees]

ADU_I2C - UNB-0, BN-1, ADU-AB: Read temperature = 44 [degrees]

ADU_I2C - UNB-0, BN-2, ADU-AB: Read temperature = 45 [degrees]

ADU_I2C - UNB-0, BN-3, ADU-AB: Read temperature = 44 [degrees]

PPS - UNB-0, BN-0: read_ppsh_stable = OK

PPS - UNB-0, BN-1: read_ppsh_stable = OK

PPS - UNB-0, BN-2: read_ppsh_stable = OK

PPS - UNB-0, BN-3: read_ppsh_stable = OK

>>> Clock-CW DCs:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	-0.42166	0.06745	-0.12695	-0.68359	0.55664	900	143	11	1	
CH4	-0.40579	0.08039	-0.08496	-0.64941	0.56445	952	151	9		
CH8	-0.74735	0.07549	-0.50098	-1.04102	0.54004	958	133	8		
CH12	0.06128	0.07991	0.38184	-0.21875	0.60059	949	140	15	1	
CH1	-0.59847	0.08563	-0.30176	-0.88379	0.58203	937	141	12		
CH5	-0.98074	0.07399	-0.69434	-1.22949	0.53516	857	170	16		
CH9	-0.73414	0.06548	-0.48340	-0.95898	0.47559	930	143	12		
CH13	-0.83064	0.07483	-0.49414	-1.04102	0.54688	943	127	13	1	

>>> Clock-CW phases:

Channel	Mean	Std	Max	Min	Diff	Nof>1U	Nof>2U	Nof>3U	Nof>4U	Nof>5U
CH0	4.32474	0.00365	4.33943	4.31112	0.02831					
CH4	6.05196	0.20631	6.07471	2.05571	4.01900	8	8	8		
CH8	6.15314	0.00419	6.16635	6.14143	0.02492					
CH12	5.61170	0.16321	5.63089	1.61028	4.02060	5	5	5	1	
CH1	4.38938	0.00429	4.40402	4.37660	0.02742					
CH5	5.96139	0.20626	5.98090	1.96564	4.01526	8	8	8		
CH9	5.83563	0.00327	5.84836	5.82575	0.02261					
CH13	5.83409	0.16320	5.85251	1.83114	4.02137	5	5	5	1	

>>> Clock-CW amplitudes:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	: 50.34319	0.09487	50.73568	50.01909	0.71659	937	126	15	2	
CH4	: 50.20358	0.12725	50.57093	49.73871	0.83222	1013	109	3		
CH8	: 49.97079	0.10461	50.25407	49.61202	0.64205	965	127	10		
CH12	: 48.97348	0.12000	49.34748	48.58651	0.76098	987	133	4		
CH1	: 49.56749	0.11922	49.90042	49.15701	0.74341	953	136	7		
CH5	: 51.83209	0.10727	52.15844	51.47277	0.68566	949	138	8		
CH9	: 50.77951	0.09822	51.06158	50.48255	0.57902	990	124	2		
CH13	: 49.36934	0.11015	49.77744	49.02657	0.75087	955	129	9		

>>> Clock-CW noise peaks:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	: 1.19086	0.10103	1.74160	0.75911	0.98249	781	154	40	7	2
CH4	: 1.18496	0.12818	1.68664	0.62256	1.06409	739	196	42	1	
CH8	: 1.27158	0.13843	1.77412	0.79933	0.97479	947	153	5		
CH12	: 1.02602	0.13047	1.58846	0.67570	0.91276	936	141	14	1	
CH1	: 1.12587	0.19398	1.64108	0.60528	1.03580	1128	65			
CH5	: 1.11056	0.17152	1.95001	0.61687	1.33314	990	107	8	1	
CH9	: 1.15135	0.12463	1.47293	0.66588	0.80706	1073	92	1		
CH13	: 1.13606	0.09286	1.49225	0.77115	0.72110	877	152	22		

>>> Clock-CW SNRs:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	: 48.31803	1.00302	51.93488	44.83623	7.09865	925	150	11		
CH4	: 47.31698	1.03429	52.03864	44.24759	7.79105	876	149	31	4	
CH8	: 46.41384	0.66143	49.08686	44.18999	4.89687	970	122	11	1	
CH12	: 47.28212	1.06900	52.57789	44.61828	7.95961	825	145	39	3	
CH1	: 46.67066	0.55985	49.25505	44.67974	4.57531	876	148	16	1	
CH5	: 47.70795	1.23518	52.20200	45.05191	7.15009	947	130	17		
CH9	: 48.29352	0.65363	51.12801	45.97680	5.15121	917	144	8	1	
CH13	: 47.75528	1.09545	52.22555	44.65296	7.57258	965	145	8	2	

>>> Test case result: PASSED (run time 6505 s)

8.3 SP 0, 2, 16, 18, 32, 34, 48, 50

Results of Figure 8, Figure 11, Figure 14, Figure 17 and Figure 20.

[2012:12:12 11:25:48] - (3) TC_BN_CAPTURE_AND_PLOT -

>>> Title : Test case to plot the ADC-[ABCD] data for UNB-[0, 1, 2, 3], BN-[0], SP-[0, 2]:

Rep-0

Rep-600

Rep-1200

Rep-1800

Rep-2400

Rep-2999

SENS - UNB-0, BN-0: FPGA temperature = 36 [degrees]

SENS - UNB-1, BN-0: FPGA temperature = 35 [degrees]

SENS - UNB-2, BN-0: FPGA temperature = 37 [degrees]

SENS - UNB-3, BN-0: FPGA temperature = 34 [degrees]

ADU_I2C - UNB-0, BN-0, ADU-AB: Read temperature = 42 [degrees]

ADU_I2C - UNB-1, BN-0, ADU-AB: Read temperature = 45 [degrees]

ADU_I2C - UNB-2, BN-0, ADU-AB: Read temperature = 47 [degrees]

ADU_I2C - UNB-3, BN-0, ADU-AB: Read temperature = 46 [degrees]

ADU_I2C - UNB-0, BN-0, ADU-CD: Read temperature = 45 [degrees]

ADU_I2C - UNB-1, BN-0, ADU-CD: Read temperature = 45 [degrees]

ADU_I2C - UNB-2, BN-0, ADU-CD: Read temperature = 48 [degrees]

ADU_I2C - UNB-3, BN-0, ADU-CD: Read temperature = 44 [degrees]

PPS - UNB-0, BN-0: read_ppsh_stable = OK

PPS - UNB-1, BN-0: read_ppsh_stable = OK

PPS - UNB-2, BN-0: read_ppsh_stable = OK

PPS - UNB-3, BN-0: read_ppsh_stable = OK

>>> Clock-CW DCs:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	-0.42577	0.06896	-0.14648	-0.69727	0.55078	922	137	18	2	
CH16	-0.69219	0.07422	-0.44727	-0.97656	0.52930	988	118	9		
CH32	-0.40899	0.07613	-0.16602	-0.70703	0.54102	956	141	8		
CH48	-0.43202	0.06049	-0.20508	-0.64453	0.43945	904	150	14		
CH2	-0.05317	0.08180	0.22266	-0.37988	0.60254	882	166	18		
CH18	-0.99522	0.07741	-0.73828	-1.27344	0.53516	930	140	8		
CH34	-0.63700	0.05879	-0.44629	-0.85352	0.40723	958	141	10		
CH50	-0.37889	0.06706	-0.12402	-0.63281	0.50879	948	146	15		

>>> Clock-CW phases:

Channel	Mean	Std	Max	Min	Diff	Nof>1U	Nof>2U	Nof>3U	Nof>4U	Nof>5U
CH0	4.32232	0.07313	4.33899	0.32277	4.01623	1	1	1		
CH16	4.75675	0.00400	4.77385	4.74367	0.03018					
CH32	4.71390	0.00449	4.73176	4.70050	0.03125					
CH48	5.27260	0.00317	5.28424	5.25292	0.03132					
CH2	4.86304	0.07305	4.87637	0.86919	4.00718	1	1	1		
CH18	5.07039	0.00537	5.08685	5.05363	0.03322					
CH34	4.96372	0.00343	4.97590	4.94757	0.02833					
CH50	4.69188	0.00451	4.70595	4.67605	0.02990					

>>> Clock-CW amplitudes:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	: 50.34196	0.09420	50.79154	49.93703	0.85451	937	139	10	3	
CH16	: 45.49816	0.13811	45.92758	45.09665	0.83093	1102	77	1		
CH32	: 46.33243	0.11063	46.67618	45.89428	0.78190	939	143	10		
CH48	: 45.57226	0.14221	46.00609	45.19072	0.81537	1049	90	2		
CH2	: 48.88303	0.18579	49.25254	48.24772	1.00482	1132	74	5		
CH18	: 45.97415	0.11086	46.44006	45.62167	0.81839	1016	113	6	1	
CH34	: 46.46270	0.06692	46.78317	46.26832	0.51485	954	141	7	2	
CH50	: 45.67884	0.09150	45.96296	45.24268	0.72027	921	149	10	2	

>>> Clock-CW noise peaks:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	: 1.18490	0.09738	1.82447	0.74560	1.07888	786	135	41	12	2
CH16	: 1.07799	0.11703	1.69871	0.71594	0.98277	777	155	37	9	1
CH32	: 1.12134	0.14826	1.66092	0.65529	1.00563	931	160	3		
CH48	: 1.18854	0.09665	1.56249	0.78805	0.77444	885	143	17	1	
CH2	: 1.13225	0.13454	1.72437	0.64760	1.07677	849	150	27	2	
CH18	: 1.15346	0.17133	1.83011	0.68263	1.14749	901	179	9		
CH34	: 1.18868	0.09997	1.76560	0.73559	1.03001	790	156	37	4	1
CH50	: 1.08560	0.13703	1.55193	0.73914	0.81279	962	129	12		

>>> Clock-CW SNRs:

Channel	Mean	Std	Max	Min	Diff	Nof>1S	Nof>2S	Nof>3S	Nof>4S	Nof>5S
CH0	: 48.24802	1.02914	51.49920	45.01627	6.48293	958	146	6		
CH16	: 47.99723	1.02514	53.44261	44.93290	8.50971	725	165	46	9	2
CH32	: 47.31024	1.10166	52.37393	44.91421	7.45972	996	110	15	2	
CH48	: 48.72019	1.24740	52.92224	44.96558	7.95667	946	143	7		
CH2	: 46.82169	1.39027	52.93042	43.71416	9.21626	913	136	26	3	
CH18	: 47.20291	0.89450	53.14993	44.61008	8.53985	663	163	59	20	9
CH34	: 48.30816	0.68355	50.65132	45.10364	5.54768	864	150	20	4	
CH50	: 47.57653	0.56884	50.74996	45.73132	5.01865	780	151	38	12	2

>>> Test case result: PASSED (run time 9756 s)