LBT PROJECT

ARGOS BCU mini-crate SYSTEM DESIGN

Document :	LBT-MIC-TRE-00910-0001	
Issue :	3	
Date :	31.01.2014	
Prepared by :	MICROGATE	
	M.Andrighettoni	.....
	R.Biasi	.....
Checked by :	R.Biasi	.....
Approved by :	R.Biasi	.....
Released by :	S.Rizzetto	.....

MICROGATE S.r.l.
Via Stradivari, 4
I 39100 BOLZANO
www.microgate.it

CHANGE RECORDS

ISSUE	DATE	Author	Approved	QA/QC	SECTION / PARAG. AFFECTED	REASON/INITIATION DOCUMENTS/REMARKS
draft	02.05.2011	M. Andrighettoni			All	First draft version
1	13.05.2011	M. Andrighettoni			8.4.4, 8.4.6, 8.4.7	First release – updated paragraphs 8.4.4, 8.4.6 and added paragraph 8.4.7
2	27.11.2012	M. Andrighettoni			6	Second release, included information for new DSP-ADC board
3	31.01.2014	M. Andrighettoni			7.3, 8.2, 8.2.4.7 and 10	- updated Table 1 - updated paragraph 7.3 - some integration on paragraph 8.2 - added alternative centroid algorithm 8.2.4.7 - updated chapter 10

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 ACRONYMS

ADC	Analog to Digital Converter
AO	Adaptive Optics
BCU	Microgate Basic Computational Unit board
CCD	Charge Coupled Device
DM	Deformable Mirror
DSP	Digital Signal Processor
DSP-ADC	Dedicated DSP board with 8x high speed ADC
EOF	End Of File
FLAO	First Light Adaptive Optics
FPGA	Field Programmable Gate Array
HP	Width unit for 19” chassis, corresponding to 0.2” (5.08mm)
HV	High Voltage
HVC	Microgate High Voltage Control board
LBT	Large Binocular Telescope
LBTC	Large Binocular Telescope Corporation
LGS	Laser Guide Star
LVDS	Low Voltage Differential Signaling
MPIfR	Max-Planck-Institut für Radioastronomie
Na	Sodium Laser Guide Star
NaBCU	BCU slope computer for the Sodium Laser Guide Star
NGS	Natural Guide Star
PCB	Printed Circuit Board
pnCCD	Very fast high resolution detector developed by MP-HLL
SDRAM	Synchronous Dynamic Random Access Memory
TBC	To Be Confirmed
TBD	To Be Defined
TE	Width unit for 19” chassis, corresponding to 0.2” (5.08mm)
TT	Tip-tilt
U	Height unit for 19” chassis, corresponding to 1.75” (44.45mm)
WFS	Wavefront Sensor
w.r.t.	with respect to

2 APPLICABLE DOCUMENTS

[AD1] Gilles Orban de Xivry, "Requirements for the ARGOS BCU", doc. "ARGOS_BCU_v1.3", Revision 1.3 dated 08.09.2010.

[AD2] SIS GmbH, hllframegen-M-0-1-v002.pdf, "HLL Framegenerator User Manual", Revision 0.02 dated 05.06.09

[AD3] Minute of meeting (telecon) held on 05.05.2010, doc. ARGOS-BCU-MOM05052010.doc

[AD4] Matthias Drochner et al., "Implementation of a Gigabit-Link Protocol with the PLX PCI 9054 Interface", doc. " gigalinkenglish_for_microgate.pdf", Version 1.12

[AD5] J.Ziegleder et al, "ARGOS - Advanced Rayleigh Ground layer adaptive Optics System", doc. "argos_pdr_wfs_electronics_081222jz.doc", Issue 1.0 dated 05.11.2008

[AD6] "Fast pnCCD overview", doc. Overview_081213

[AD7] Marcel Elberich, RS485 Tip-tilt communication protocol, doc. "argos_transmission_package-1.pdf" dated 04.08.2010

3 REFERENCE DOCUMENTS

[RD1] Microgate, DSP firmware spread sheet, xls. "ARGOS_CodesMemMap_v1.xls" dated 02.05.2011

4 INTRODUCTION

This document describes the design of the BCU mini-crate for the ARGOS system and all information needed to initialize and to use the BCU mini-crate.

The information provided are complementary to the system initialization Matlab script that is provided together with the project data package. This script is an effective guideline for the final system implementation, together with the details provided by this document.

5 SYSTEM DESIGN

According to [AD1] the BCU mini-crate has been integrated as shown in Figure 1. The main elements in the integrated hardware are:

- one BCU board with the following the interfaces:
  - one 1Gbit Ethernet copper link, with standard RJ45 connector, connected to the ARGOS supervisor
  - four Fiber Channel optical links, with standard pair LC connector, the first two links are used for the connection with the pnCCD, the third for the connection with the either FLAO or, in a subsequent project phase, with the NaBCU and the fourth for the connection with the secondary mirror.
  - one RS485 serial link, with standard Fischer 7 poles connector (see 12.1 for the pin-out scheme), connected to the MPIfR device for TT slope computation
- one 3rd generation DSP board (called DSP-ADC 8ch) equipped with two ADSP TS201 microcontrollers, used to satisfy the high computational performances requested by ARGOS. This new board also includes the analog to digital electronics needed for the direct acquisition of the pnCCD analog output.
- two HVC boards used to drive the three piezo TT mirrors for the field pointing of the three LGS of WFS. Each board has 6 strain gauge inputs and 6 high voltage outputs, so the first board drives two mirrors (3+3 inputs and outputs) and the second the third mirror, three inputs and outputs are not used.

Figure 1 - LGS BCU integrated mini-crate

The

Figure 2 is the design block of the ARGOS BCU mini-crate, called LGS BCU, with all the interconnection and data flux involved in the real time computation.

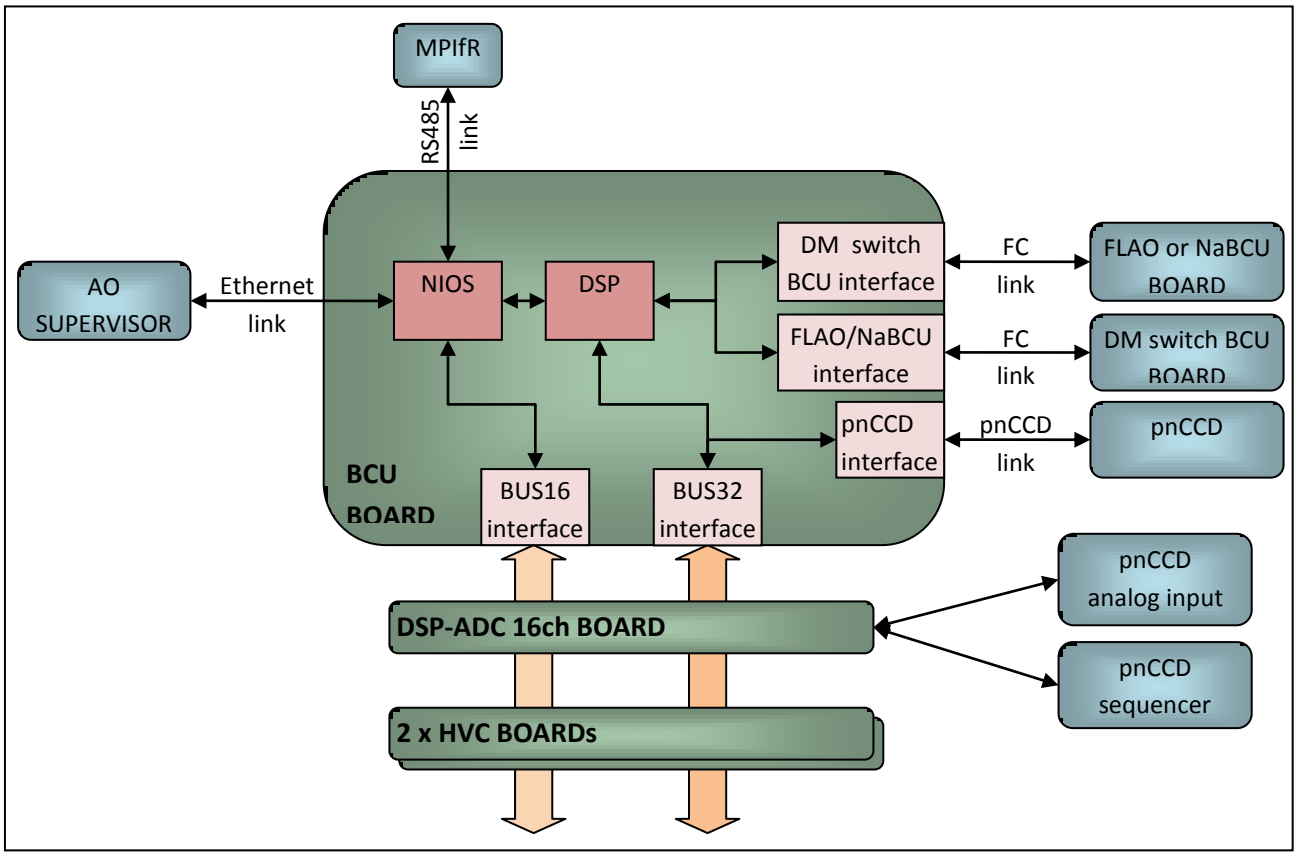

Figure 2 - LGB BCU design scheme

The main elements and the data flux summarized in the design scheme are:

- the arbiter of the entire real time system is the BCU board.
- the ARGOS supervisor is connected with BCU mini-crate using a standard 1Gbit Ethernet link, this link is used for the system initialization, including firmware download, parameters configuration and link + computation enabling. When the system is running the same link is used for the diagnostic data storing, acquisition and monitoring.
- The APD counts and the TT slopes acquired and computed by MPIfR are sent to the BCU using a RS485 link, which protocol is defined in [AD7]. The serial data are temporary stored to a local buffer until the <cr> character arrives indicating an end of packet (EOF). Once the EOF is detected the validity checks are done and in case of no errors, the packet is passed to the BCU-DSP to be included in the slope vector for the DM. There is a dedicated mechanism to ensure that the data transfer to the slope vector is executed in atomic mode.
- The optional FLAO and Na slope vector is connected with the BCU mini-crate using the third Fiber Channel link, if a new set of slope arrives to the ARGOS system it is copied to the BCU-DSP memory

with a dedicated mechanism to ensure the atomicity of the upload. The new data set will included in the DM slope vector and passed to the DM together to the next pnCCD slope vector.

- The pnCCD interface is a dedicated module directly linked to the pnCCD controller according to the protocol defined in [AD1]. After the introduction of the new DSP-ADC control board this input is an optional input with respect to the direct acquisition of the analog pnCCD outputs using the on board ADC devices. On both cases the received frame is copied to the DSPs of the DSP-ADC board memory and the slope computation proceed always in the same way, regardless of the adopted interface; the computation is done in parallel to the pixel download in order to minimize the slope computational time lag.
- Once the computation is completed, the BCU-DSP reads back the new slope vector and collects the new data with the MPIfR TT slopes and, when applicable, the FLAO and Na slopes. The entire vector is prepared according to the DM Switch BCU standard input and sent to the Switch BCU using the fourth Fiber Channel link to start a DM command update mechanism.
- After completion of the DM command update, the system creates and saves the diagnostic record according to [AD1]. The data is stored in the BCU SDRAM bulk memory and, if enabled, downloaded to the ARGOS supervisor using an automatic and very efficient mechanism (called master-BCU). The pnCCD pixel frames are stored to the DSP-ADC SDRAM bulk memory, where up to 960 frames at full rate can be stored; the full rate data can be read later, stopping the frame storing and simply reading the DSP-ADC SDRAM memory using the Ethernet link and we recommend to avoid the reading while the system is still running, on the contrary in real time mode a decimated pixel frame (up to 50 Hz) can be downloaded to the ARGOS supervisor.

5.1 Computational flux description

As general overview Figure 3 shows the activity sequence of the various elements of the mini-crate BCU system and how the activity are linked together when a new pnCCD frame arrives to the BCU fast link interface. The duration of each block is not proportional to the actual value, it is just a conceptual timing diagram.

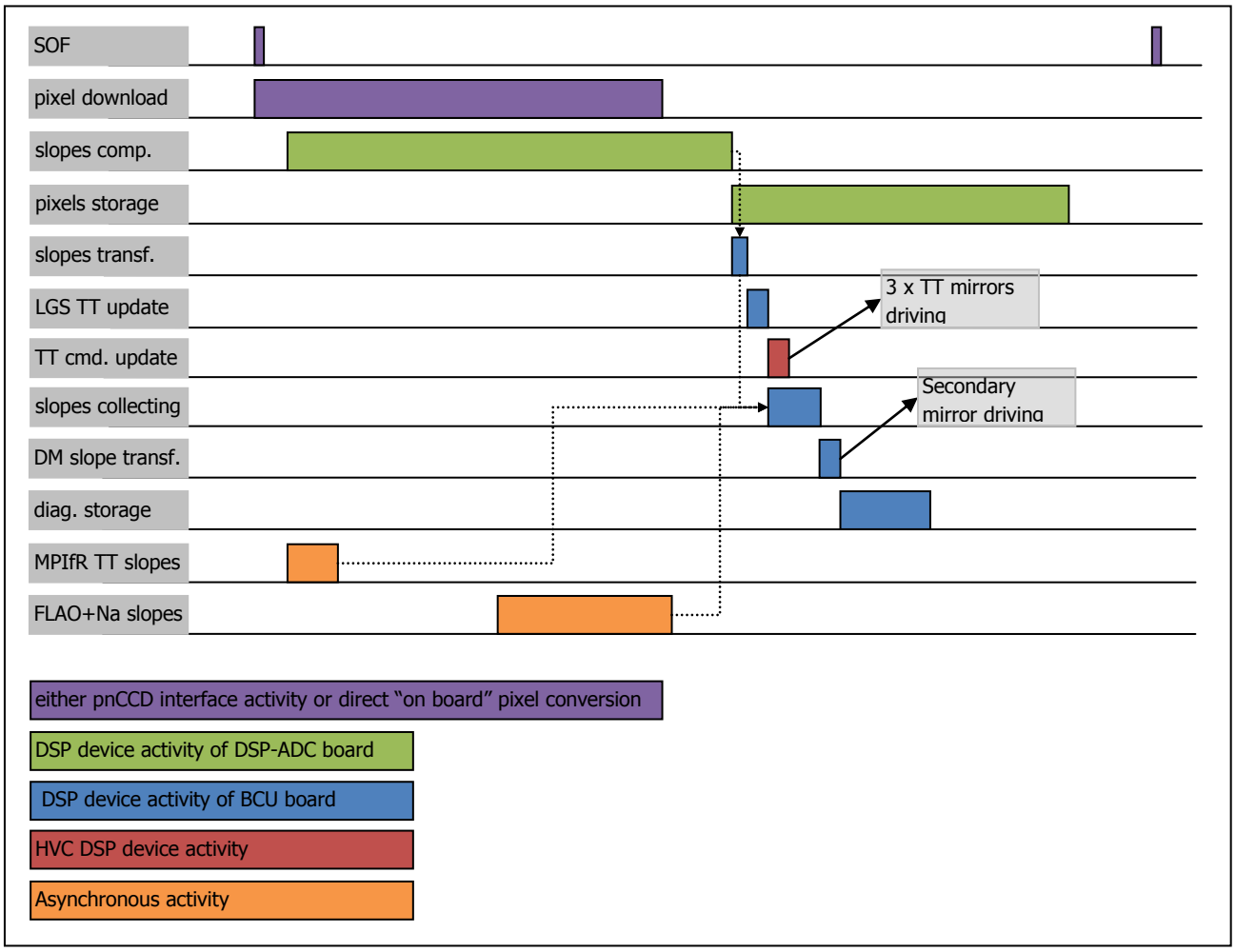

Figure 3 – Computationa sequence diagram

6 DSP-ADC 8ch board

The ARGOS mini-crate is powered with a new ADC board specifically designed to directly acquire the analog output of the pnCCD, amplify the signals, convert to digital values and finally write to the DSP memory for the slope computation.

Figure 4 – The new DSP-ADC 8ch board

The boards includes 8 fully independent differential input channels, each comprehending an input filter and amplifier that feeds a high speed analog to digital converter.

The input stage is based on the LTC6404 differential input/output operational amplifier from Linear Technology. The differential input is terminated on-board to guarantee 100ohm input impedance; the signal is first reduced in amplitude, then amplified and filtered by the differential operational amplifier. The input passive amplitude reduction is required to guarantee that the operational amplifiers operates in its stable region. The amplifier implements two real poles at 168MHz and 44Mhz respectively, with a total τ of 4.4ns. This leads to a 16^{th} bit settling within 48ns; the acquisition timing can be tuned by properly setting the delay between ADC clock and acquisition. An additional feature of the input stage is the programmable offset level: the offset can be programmed by SW, independently on all channels, from 0.6V to 1.92V, to adapt it to different input configurations. In the specific pnCCD case the input has been set at 1.1V.

The analog to digital conversion stage is based on the LTC2192 16bit ADC from Linear Technology. Each device comprehends two independent ADCs, so four identical devices are installed on the board to serve the 8 input channels. The device operates up to 65 MSamples/s, but in the current configuration data are sampled at 12.5 MSamples/s. In the development phase we have evaluated the possibility of 2x oversampling the input signal and averaging the acquisitions to improve the SNR, but this proved to be

quite ineffective, as explained later in this section. The digital interface is based on a 2-lanes per channel high speed LVDS interface, connecting the ADC directly to the FPGA, that forwards the incoming data to the two DSP devices.

The ADC acquisition is controlled by the input lines arriving from the Sequencer board. This board provides a clock @50MHz for the ADC sampling, a pixel enable @12.5 Mhz to indicate which ADC acquisitions are valid and a Start of Frame signal to synchronize the pnCCD acquisition with the first valid pixel. In Figure 5 there is an example of the signal generated by the Sequencer board, the red trace is the pixel clock, the yellow trace is the start of frame signal and the green and blue traces are the differential analog signals of one channel.

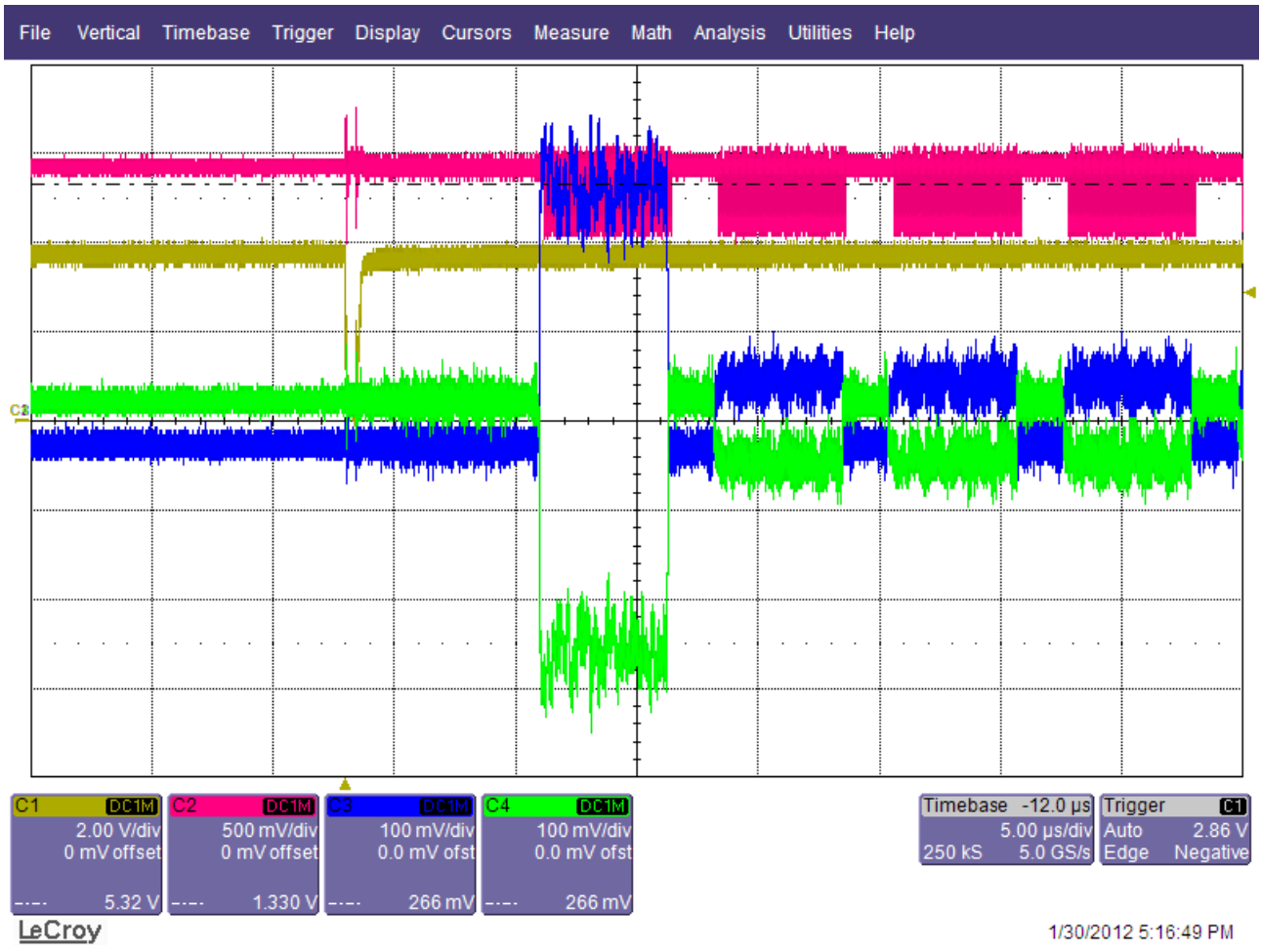

Figure 5 – Capture of input signal from Sequencer board

The Figure 6 shows in detail the pixel clock duration (red trace) with respect to the analog input (green and blue traces in differential). Considering that the ADC clock runs at 50MHz with respect to the pixel clock

that runs at 12.5MHz it means that for each pixel we have four ADC acquisitions. The two acquisitions taken when the pixel clock is low must be discarded because on this period there is the pixel value transition while the two acquisitions taken during the high phase of pixel clock can be considered valid. Considering this we can choose between two options for the pixel acquisition:

- we take the two valid pixel acquisitions (during the high phase of pixel clock) and then compute an averaged value to reduce the analog noise;
- increment the low pass filter gain as much as possible of the analog input pixel value and reduce the ADC sampling rate from 50MHz to 12.5MHz acquiring the pixel value in the steady state of the filtered analog input.

The DSP-ADC board is able to implement both options but to switch between them requires a different FPGA configuration and analog low pass filter configuration. Currently the delivered ADC-DSP boards for the ARGOS projects implements the second acquisition solution.

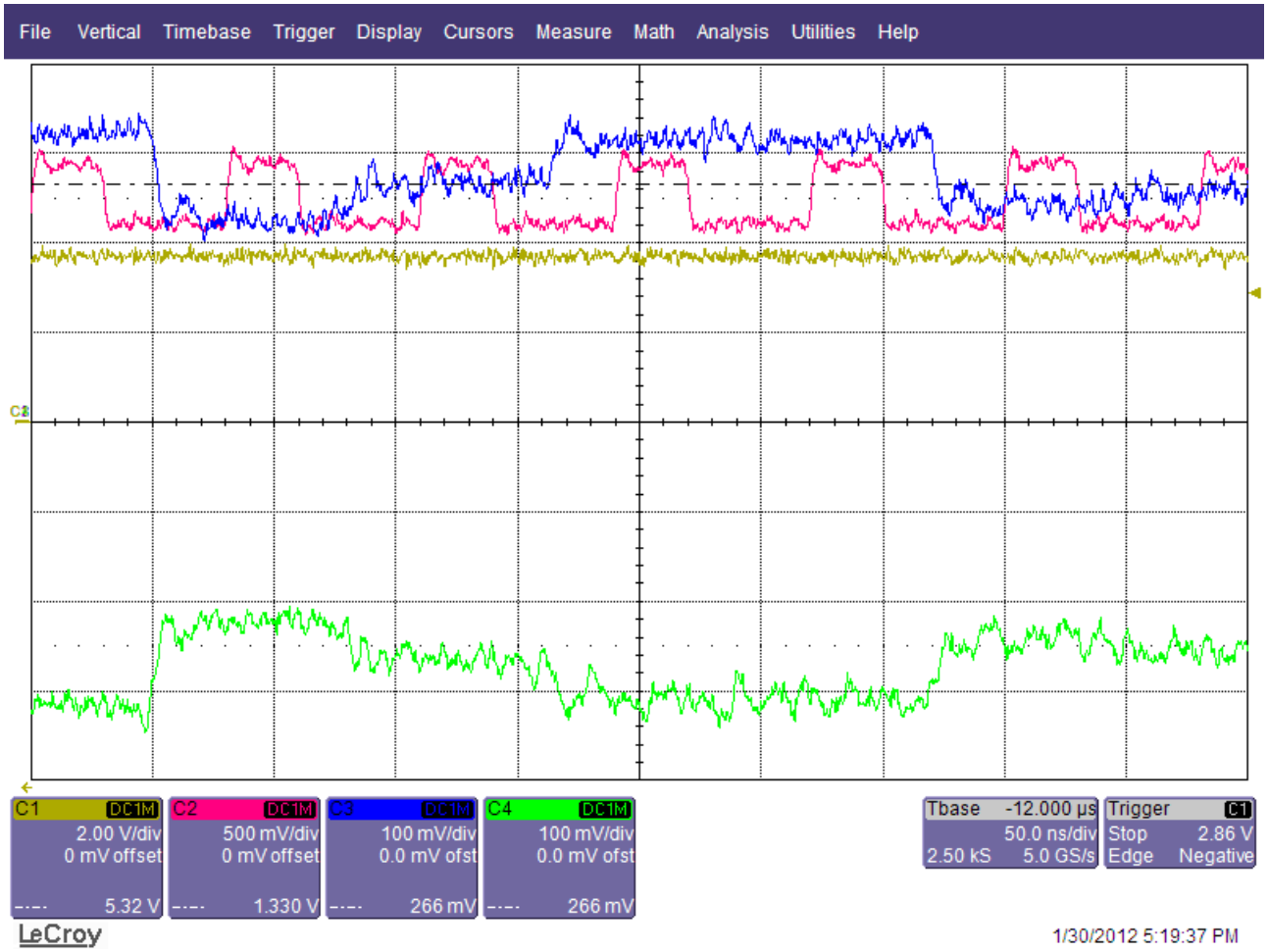

Figure 6 – Detail of the pixel clock with respect to the analog input

Finally, to minimize the noise the ADC conversion shall be well tuned to latch the analog input at the proper time. The tuning is configurable at FPGA level only. The ADC-DSP boards delivered for the Argos project are already well tuned as shown in Figure 7 where the orange trace is the analog input (filtered) and the blue trace is the ADC clock (the sampling is done on the rising edge of the clock signal).

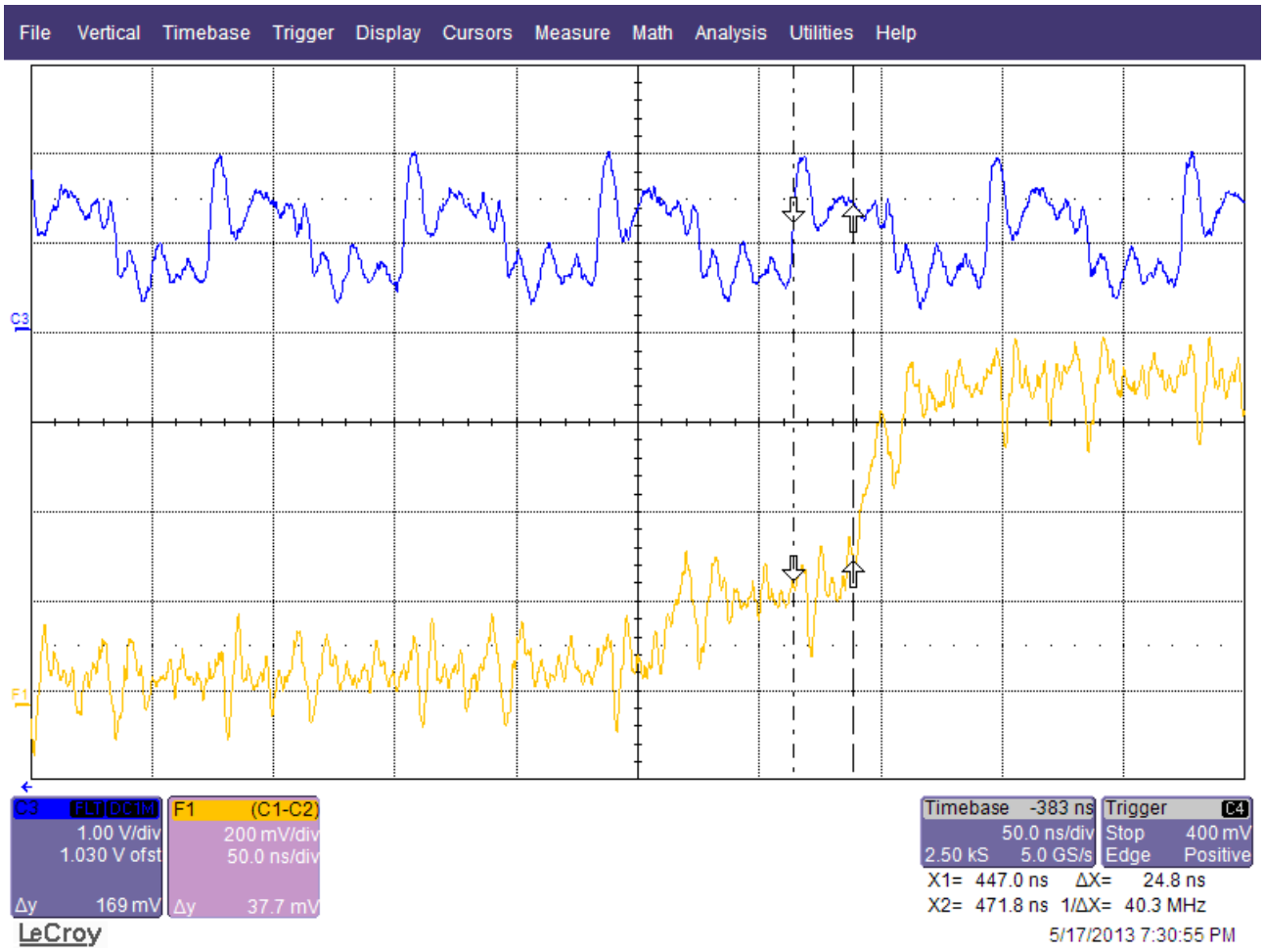

Figure 7 – ADC sampling time

7 LGS BCU MINI-CRATE FIRMWARE DESCRIPTION

In this chapter we describe the whole firmware implemented in the BCU mini-crate.

7.1 Firmware configuration

Several different pieces of firmware run on the BCU Mini-Crate system, in particular each board has an FPGA firmware, a NIOS firmware and a DSP firmware. The first two firmware (FPGA and NIOS) are factory downloaded to the embedded flash and at the system booting the firmware is automatically uploaded and launched. The update of this firmware might be necessary in case of bugs or system improvements; the new firmware can be uploaded on the on-board flash remotely (not necessary by Microgate) using a dedicated Ethernet procedure. This intervention is not dangerous for the system but should done with attention; we recommend to perform it with the support of Microgate personnel.

Conversely, the DSP firmware is downloaded to the DSP controllers at every system restart; the DSP executable file (.ldr extension) is part of the delivery packet; the DSP firmware downloading mechanism is described in 7.2.

The present document refers to the firmware releases listed in Table 1.

board	FPGA firmware	NIOS firmware	DSP firmware (name and version)
BCU	ver. 11.71	ver. 5.08.0046	BCUSlopeComputer, ver. 6.08
DSP-ADC	ver. 8.46	ver. 5.06.0040	DSPSlopeComputer, ver. 2.04
HVC	ver. 2.02	ver. 4.02.0020	HVCMainProgram, ver. 3.00

Table 1 - BCU mini-crate firmware releases

7.2 DSP code uploading

All the DSP devices of the ARGOS BCU mini-crate at the start-up and after a reset are in idle condition. Then the first step of the system initialization is the DSP firmware upload, then the control variables initialization and finally the start of the code.

The firmware code can uploaded into the DSP by means of standard 'Write sequential' commands. The code is sent in fragments whose size does not exceed the maximum allowable Ethernet packed size.

7.2.1 DSP loader file data format

The DSP code is stored in '.ldr' files delivered as part of the data pack of the ARGOS BCU system. This is an ASCII file, structured as a vector of 32 bit words in hex format (0xXXXXXXXX).

The file contains the following fields:

Field description	Content	Length (32bit words)
Start of data memory vector. The first 28 bits contain the data length in 32 bit words.	0x4XXXXXXX	1
First address of data memory vector.	Start address	1
Data	Data	XXXXXXX
Start of vector to be zero-filled. The first 28 bits contain the vector length in 32 bit words.	0x8XXXXXXX	1
First address of zero-filled vector. The vector of zeros shall be sent generated and sent according to the length and start address information.	Start address	1
Start of program data. Program starts from location 0x00.	0x00000000	256

Table 2 – DSP loader file format

7.2.2 DSP code starting

After sending the code (it might be recommendable to check the code by reading if back and comparing the memory content), the DSP is still idling. To start operation, write **0 (zero)** to the DSP memory location **0x180730**.

7.3 DSP memory map

Before to enter in the detail of the system parameters initialization it is important to understand the adopted approach used to implement the DSP code. In fact, in order to optimize the real time computation all the ARGOS slopes algorithm and memory allocation has been implemented using a memory “pointer” approach. In the code we have three kinds of variables:

1. the shared memory area where the input or output data are copied typically using a DMA mechanism;
2. control registers variables used to set in the code the vector dimensions, loops etc.;
3. the pointers to the vectors (or matrices) of the parameters, using this approach the memory allocation of the parameters and of the input/output vectors is not fixed and can be modified by the user.

This approach has the advantage to have, for example, a code completely flexible in terms of number of “common mode”, “contour” or “slope” pixels, number of sub-apertures etc. The drawback is that the system initialization must be done taking care of the DSP memory and that all the DSP registers must be

correctly initialized to avoid memory overlapping or leaking, in fact no memory integrity checks are executed by the DSP.

To give an help to the user, an Excel spread sheet has been realized, in this Excel file there is a minimal set of input parameters to be initialized by the user; then, the spreadsheet optimizes automatically the DSP memory allocation of all three DSP firmware and returns how to initialize the DSP memory. The Excel file is named *ARGOS_CodesMemMap_v1.xls* see [RD1].

7.3.1 DSP memory map of BCU-DSP board

The DSP micro controller mounted on the BCU board is an ADSP TS101, it is equipped with three independent memory banks of 65536 dwords at 32bit each. The typical memory organization is to put the binary code at the beginning of the bank #0 in a fixed area of 4096 dwords.

After this area there is a DSP registers area placed on fixed locations; this area is used as shared memory with other external devices and for this reason it is located area in fixed positions.

Subsequent to the fixed area there are all DSP single variables used in the code for the real time computation, including the vector dimension and the memory pointers to the data vectors, in this area there are also the processing or error counters and a read only area for internal use. In general all these variables can be divided in “control variables” that should be initialized before starting the DSP code, and the “internal variables” that are read-only and should NEVER be overwritten. For a detailed description of all DSP variables see [RD1] - sheet *BCUSlopeComputation*.

7.3.2 DSP memory map of DSP-ADC board

The DSP-ADC board mounts two ADSP TS201 controllers, more powerful respect to the ADSP TS101, they are equipped with twelve independent memory banks of 65536 dwords at 32bit each. The typical memory organization is to put the binary code in the first bank (called bank #0A) in a fixed area of 4096 dwords.

In the second bank (bank #0B) there is a fixed located DSP registers area, this area is used as shared memory with other external devices and for this reason it is a fixed allocated area.

Subsequent to the fixed area there are all DSP single variables used in the code for the real time computation, including the vector dimensions and the memory pointers to the data vectors, in this area there are also the processing or error counters and a read only area for internal use. In general all these variables can be divided in “control variable” that should be initialized before starting the DSP code and the “internal variables” that are read only and should NEVER be overwritten. For a detailed description of all DSP variables see [RD1] - sheet *DSPSlopeComputation*.

7.3.3 DSP memory map of HVC-DSP board

The HVC board mounts one ADSP TS101 controller, same of the BCU-DSP device.

Similar to the BCU-DSP memory, it is organized putting the binary code at the beginning of the bank #0 in a fixed area of 4096 dwords.

After this area there is a fixed located DSP registers area, this area is used as shared memory with other external devices and for this reason it is a fixed allocated area.

Subsequent to the fixed area there are all DSP single variables used in the code for the real time computation, including the vector dimensions and the memory pointers to the data vectors, in this area there are also the processing or error counters and a read only area for internal use. In general all these variables can be divided in "control variables" that should be initialized before starting the DSP code, and the "internal variables" that are read-only and should NEVER be overwritten. For a detailed description of all DSP variables see [RD1] - sheet *HVCMainProgram*.

7.4 DSP variables map and description

7.4.1 BCU-DSP variables

Table 3 shows all the variables that should be initialized in the *BCUSlopeComputer* code. For the proper initialization and memory locations see [RD1] - sheet *BCUSlopeComputation*.

variable name	type	description
_bscub_DiagnosticFrameDec	uint32	Diagnostic frame decimation, see Table 52
_bscub_NumSlopesLGS0	uint32	number of slopes of the LGS #0 according to [RD1]
_bscub_NumSlopesLGS1	uint32	number of slopes of the LGS #1 according to [RD1]
_bscub_NumSlopesLGS2	uint32	number of slopes of the LGS #1 according to [RD1]
_bscfb_InvNumSlopesLGS0	float	reciprocal of number of sub-apertures of the LGS #0
_bscfb_InvNumSlopesLGS1	float	reciprocal of number of sub-apertures of the LGS #1
_bscfb_InvNumSlopesLGS2	float	reciprocal of number of sub-apertures of the LGS #2
_bscub_FlaoNaNumSlopes	uint32	number of slopes of the FLAO/Na detectors according to [RD1]
_bscub_NumFLTimeout	uint32	Counter register with the total fast link communication timeout errors
_bscub_NumFLCrcErr	uint32	Counter register with the total fast link communication CRC errors
_bscub_ReplyVectorPtr	uint32	Pointer to the area of the retrieved data from the DSP-ADC board
_bscub_ExtFlaoNaSlopeVectPtr	uint32	Shared memory with the FLAO/Na slope vector received from FLAO or Na BCU
_bscub_ExtFlaoNaStartRTRPtr	uint32	Shared memory with the start trigger area for the FLAO/Na slope vector
_bscub_RemapSlopeVectorPtr	uint32	Pointer to the area for the remapping of the retrieved slope vectors from the DSP-ADC board to the final slope vector to send to the DM
_bscub_HeaderDiagPtr	uint32	Pointer to the area of the header of the diagnostic record
_bscub_SlopeVectorPtr	uint32	Pointer to the area of the final pnCCD slope vector to send to the DM
_bscub_RotTTSlopeVectPtr	uint32	Pointer to the area of the TT slope vector computed by the MPIfR system to send to the DM
_bscub_FlaoNaSlopeVectPtr	uint32	Pointer to the area of the FLAO/Na slope vector to send to the DM

_bscub_StartRTRPtr	uint32	Footer of the slope vector with the start DM reconstructor update trigger.
_bscub_TTSlopesVectorPtr	uint32	Pointer to the area of the TT slope vector for the field pointing of the WFS
_bscub_HVCDiagnosticPtr	uint32	Pointer to the area of the HVC diagnostic data
_bscub_APDCountersPtr	uint32	Pointer to the area of the APD raw counts received from the MPIfR system
_bscub_FCVectorPtr	uint32	Pointer to the area of the frames counter vector + time stamp
_bscub_FooterDiagPtr	uint32	Pointer to the area of the footee of the diagnostic record
_bscub_FastlinkCmd	uint32	Fast link command registers, see 8.4.8

Table 3 – BCU-DSP firmware variables description

7.4.2 DSP-ADC variables

Table 4 shows all the variables that should be initialized in the *DSPSlopeComputer* code. For the proper initialization and memory locations see [RD1] - sheet *DSPSlopeComputation*.

variable name	type	description
_dscub_ParamSelector	uint32	Variable to select various program options, see Table 5 for the complete description.
_dscub_HalfCCDSelection	uint32	Select the left (0) or right (1) half area of the CCD image
_dscub_LineLength	uint32	Length of CCD line (number of CCD channels of one CAMEX) according to [RD1]
_dscub_SubapSize	uint32	Sub-aperture size (number of pixels) according to [RD1]
_dscub_NumLines	uint32	Number of CCD lines of one CAMEX according to [RD1]
_dscub_NumSubaps	uint32	Number of sub-apertures processed by each DSP according to [RD1]
_dscib_CMThreshold	int32	Maximum threshold for common mode pixels
_dscub_SubapPowerCoeff	uint32	Sub-aperture power value selection: 0: n=1, 1: n=1.5
_dscub_DiagnosticFrameDec	uint32	Diagnostic frame decimation, see Table 52
_dscub_ADCBoardSelection	uint32	Selection between fiber optic interface (setting zero) and on-board ADC interface (setting one)
_dscub_CentroidAverage	uint32	During the centroid computation select between whole LGS sub-aperture pixels flux (setting one) and single sub-aperture pixels flux (setting zero)
_dscub_PixelAreaPtr	uint32	Pointer to the shared memory area of the CCD pixels, this point can't be changed.
_dscub_PixelGainAreaPtr	uint32	Pointer to the pixel gain area
_dscub_PixelOffAreaPtr	uint32	Pointer to the pixel offset area
_dscub_SubapPixelPtrAPtr	uint32	Pointer to the area of the sub-aperture pixel pointers (first bank)
_dscub_SubapPixelPtrBPtr	uint32	Pointer to the area of the sub-aperture pixel pointers (second bank)
_dscub_NumSubapsReadyPtr	uint32	Pointer to the area of the number of sub-apertures ready to be computed according to the number of lines arrived
_dscub_SubapMaxGainPtr	uint32	Pointer to the area of proportional threshold to the maximum pixel for each sub-aperture
_dscub_SubapFixThresholdPtr	uint32	Pointer to the area of the fixed threshold for each sub-aperture

_dscub_SubapPixelArmsPtr	uint32	Pointer to the area of the pixel arms for the centroid computation
_dscub_SubapLinearCoeffPtr		Pointer to the area of the sub-aperture linearizer coefficients
_dscub_SlopeOffsetAPtr	uint32	Pointer to the slope offset coefficients (first bank)
_dscub_SlopeOffsetBPtr	uint32	Pointer to the slope offset coefficients (second bank) for atomic parameters update
_dscub_CMPixelMapPtr	uint32	Pointer to the area of the common mode pixel pointers
_dscub_LGSCentroidMapPtr	uint32	Pointer to the area to define per each sup-aperture at which LGS is referred, it is used to compute the total LGS pixel flux
_dscub_SubapPixelWeightPtr	uint32	Pointer to the area of the sub-aperture pixel weight coefficients
_dscub_PixelFloatAreaPtr	uint32	Pointer to the area of the corrected and float converted pixels
_dscub_SlopeOutputHeaderPtr	uint32	Pointer to the area of the slope header
_dscub_SlopeOutputPtr	uint32	Pointer to the area of the slope vector
_dscfb_NumSubapLGS0	float	Number of sub-apertures of LGS #0 (on the local DSP)
_dscfb_NumSubapLGS1	float	Number of sub-apertures of LGS #1 (on the local DSP)
_dscfb_NumSubapLGS2	float	Number of sub-apertures of LGS #2 (on the local DSP)

Table 4 – DSP-ADC firmware variables description

Table 5 describes the meaning of each bit of the *_dscub_ParamSelector* word.

Bit num.	Description
0	parameters block selection 0 = block #0, 1 = block #1 (relevant for ARGOS & RTR)
1	not used
2	not used
3	not used
4	not used
5	enable (1) or disable (0) the delta command calculation using the actual position of the mirror respect to the old command (relevant for RTR)
6	enable (1) or disable (0) the diagnostic data storage to the SDRAM (relevant for ARGOS & RTR) – excluding the pixel frames
7	enable (1) or disable (0) the fast-link commands used to send the slope vector to the reconstructor (relevant for ARGOS)
8	select between the MDC (0) or MCF (1) reconstructor algorithm (relevant for RTR)
9	reserved (for internal enable use only)
10	reserved (for internal enable use only)
11	enable (1) or disable (0) the DM accelerometers acquisition (relevant for RTR)
12	enable (1) or disable (0) the pixel frame transfer from the DSP-ADC memory to the BCU memory (relevant for ARGOS)
13	enable (1) or disable (0) the real-time commands to the HVC boards (relevant for ARGOS)

Table 5 – *_dscub_ParamSelector* variable description

7.4.3 HVC-DSP variables

Table 6 shows all the variables that should be initialized in the *HVCMainProgram* code. For the proper initialization and memory locations see [RD1] - sheet *HVCMainProgram*.

variable name	type	description
_hvcub_delay_DAC	uint32	variable used in the local control loop
_hvcub_EnableIsr	uint32	enabling register of the local control loop isr
_hvcfb_MinCommandTT0	float	min allowable angular command for TT0 in radian
_hvcfb_MaxCommandTT0	float	max allowable angular command for TT0 in radian
_hvcfb_MinCommandTT1	float	min allowable angular command for TT1 in radian
_hvcfb_MaxCommandTT1	float	max allowable angular command for TT1 in radian
_hvcub_ResetIntegratorTT0	uint32	set to one to reset the TT0 mirror command integrator, once done the variable is automatically reset to zero
_hvcub_ResetIntegratorTT1	uint32	set to one to reset the TT1 mirror command integrator, once done the variable is automatically reset to zero
_hvcfb_SelectionMatrixTT0	float	tip-tilt commands selection matrix for TT0
_hvcfb_RotationMatrixTT0	float	tip-tilt to angular command conversion for TT0
_hvcfb_CommandOffsetTT0	float	tip-tilt command offset for TT0
_hvcfb_FFGainTT0	float	command to direct voltage output proportional factor for TT0
_hvcfb_SelectionMatrixTT1	float	tip-tilt commands selection matrix for TT1
_hvcfb_RotationMatrixTT1	float	tip-tilt to angular command conversion for TT1
_hvcfb_CommandOffsetTT1	float	tip-tilt command offset for TT1
_hvcfb_FFGainTT1	float	command to direct voltage output proportional factor for TT1
_hvcub_TT0_delay_acc	uint32	variable used in the local control loop
_hvcub_TT0_num_samples	uint32	variable used in the local control loop
_hvcub_TT0_delay_counter	uint32	variable used in the local control loop
_hvcub_TT0_acc_counter	uint32	variable used in the local control loop
_hvcub_TT0_reset_acc	uint32	variable used in the local control loop
_hvcfb_TT0_inv_num_samples	float	variable used in the local control loop
_hvcub_TT1_delay_acc	uint32	variable used in the local control loop
_hvcub_TT1_num_samples	uint32	variable used in the local control loop
_hvcub_TT1_delay_counter	uint32	variable used in the local control loop
_hvcub_TT1_acc_counter	uint32	variable used in the local control loop
_hvcub_TT1_reset_acc	uint32	variable used in the local control loop
_hvcfb_TT1_inv_num_samples	float	variable used in the local control loop
_hvcfc_TT0_pos_command	float	variable used in the local control loop
_hvcfc_TT0_curr_command	float	variable used in the local control loop
_hvcfc_TT0_dist_average	float	variable used in the local control loop
_hvcfc_TT0_curr_average	float	variable used in the local control loop
_hvcfc_TT0_bias_command	float	variable used in the local control loop
_hvcfc_TT0_bias_current	float	variable used in the local control loop
_hvcfc_TT0_cmd_current	float	variable used in the local control loop
_hvcfc_TT0_start_preshaper_cmd	float	variable used in the local control loop
_hvcfc_TT0_final_preshaper_cmd	float	variable used in the local control loop
_hvcfc_TT0_preshaped_cmd	float	variable used in the local control loop
_hvcuc_TT0_step_ptr_preshaper_cmd	uint32	variable used in the local control loop
_hvcuc_TT0_curr_ptr_preshaper_cmd	uint32	variable used in the local control loop

_hvcfc_TT0_start_preshaper_curr	float	variable used in the local control loop
_hvcfc_TT0_final_preshaper_curr	float	variable used in the local control loop
_hvcfc_TT0_preshaped_curr	float	variable used in the local control loop
_hvcuc_TT0_step_ptr_preshaper_curr	uint32	variable used in the local control loop
_hvcuc_TT0_curr_ptr_preshaper_curr	uint32	variable used in the local control loop
_hvcfc_TT0_float_ADC_value	float	variable used in the local control loop
_hvcfc_TT0_dist_B_coeff	float	variable used in the local control loop
_hvcfc_TT0_dist_A_coeff	float	variable used in the local control loop
_hvcfc_TT0_distance	float	variable used in the local control loop
_hvcuc_TT0_delayline_ptr	uint32	variable used in the local control loop
_hvcfc_TT0_pos_current	float	variable used in the local control loop
_hvcfc_TT0_pos_pre_current	float	variable used in the local control loop
_hvcuc_TT0_control_enable	uint32	variable used in the local control loop
_hvcfc_TT0_float_DAC_value	float	variable used in the local control loop
_hvcfc_TT0_sat_DAC_value	float	variable used in the local control loop
_hvcfc_TT0_nsat_DAC_value	float	variable used in the local control loop
_hvcfc_TT0_DAC_N2A_gain	float	variable used in the local control loop
_hvcfc_TT0_DAC_A2bit_gain	float	variable used in the local control loop
_hvcfc_TT0_DAC_bit_offset	float	variable used in the local control loop
_hvcuc_TT0_uint_DAC_value	uint32	variable used in the local control loop
_hvcfc_TT0_post_loop_gain	float	variable used in the local control loop
_hvcfc_TT0_post_smoothed_gain	float	variable used in the local control loop
_hvcfc_TT0_post_smoothed_step	float	variable used in the local control loop
_hvcfc_TT0_pre_loop_gain	float	variable used in the local control loop
_hvcfc_TT0_pre_smoothed_gain	float	variable used in the local control loop
_hvcfc_TT0_pre_smoothed_step	float	variable used in the local control loop
_hvcfc_TT1_pos_command	float	variable used in the local control loop
_hvcfc_TT1_curr_command	float	variable used in the local control loop
_hvcfc_TT1_dist_average	float	variable used in the local control loop
_hvcfc_TT1_curr_average	float	variable used in the local control loop
_hvcfc_TT1_bias_command	float	variable used in the local control loop
_hvcfc_TT1_bias_current	float	variable used in the local control loop
_hvcfc_TT1_cmd_current	float	variable used in the local control loop
_hvcfc_TT1_start_preshaper_cmd	float	variable used in the local control loop
_hvcfc_TT1_final_preshaper_cmd	float	variable used in the local control loop
_hvcfc_TT1_preshaped_cmd	float	variable used in the local control loop
_hvcuc_TT1_step_ptr_preshaper_cmd	uint32	variable used in the local control loop
_hvcuc_TT1_curr_ptr_preshaper_cmd	uint32	variable used in the local control loop
_hvcfc_TT1_start_preshaper_curr	float	variable used in the local control loop
_hvcfc_TT1_final_preshaper_curr	float	variable used in the local control loop
_hvcfc_TT1_preshaped_curr	float	variable used in the local control loop
_hvcuc_TT1_step_ptr_preshaper_curr	uint32	variable used in the local control loop
_hvcuc_TT1_curr_ptr_preshaper_curr	uint32	variable used in the local control loop
_hvcfc_TT1_float_ADC_value	float	variable used in the local control loop
_hvcfc_TT1_dist_B_coeff	float	variable used in the local control loop
_hvcfc_TT1_dist_A_coeff	float	variable used in the local control loop
_hvcfc_TT1_distance	float	variable used in the local control loop

_hvcuc_TT1_delayline_ptr	uint32	variable used in the local control loop
_hvcfc_TT1_pos_current	float	variable used in the local control loop
_hvcfc_TT1_pos_pre_current	float	variable used in the local control loop
_hvcuc_TT1_control_enable	uint32	variable used in the local control loop
_hvcfc_TT1_float_DAC_value	float	variable used in the local control loop
_hvcfc_TT1_sat_DAC_value	float	variable used in the local control loop
_hvcfc_TT1_nsat_DAC_value	float	variable used in the local control loop
_hvcfc_TT1_DAC_N2A_gain	float	variable used in the local control loop
_hvcfc_TT1_DAC_A2bit_gain	float	variable used in the local control loop
_hvcfc_TT1_DAC_bit_offset	float	variable used in the local control loop
_hvcuc_TT1_uint_DAC_value	uint32	variable used in the local control loop
_hvcfc_TT1_post_loop_gain	float	variable used in the local control loop
_hvcfc_TT1_post_smoothed_gain	float	variable used in the local control loop
_hvcfc_TT1_post_smoothed_step	float	variable used in the local control loop
_hvcfc_TT1_pre_loop_gain	float	variable used in the local control loop
_hvcfc_TT1_pre_smoothed_gain	float	variable used in the local control loop
_hvcfc_TT1_pre_smoothed_step	float	variable used in the local control loop
_hvcfb_TT0_pos_coeff	float	variable used in the local control loop
_hvcfb_TT1_pos_coeff	float	variable used in the local control loop
_hvcfb_preshaper_cmd_buffer	float	variable used in the local control loop
_hvcfb_preshaper_curr_buffer	float	variable used in the local control loop

Table 6 – HVC-DSP firmware variables description

7.5 NIOS variables map and description

The NIOS firmware (called also housekeeping firmware) is the code that runs in the embedded FPGA microcontroller, the aim of this code is the board initialization, the managing of the diagnostic link (Ethernet link) and the acquisition of the diagnostic of the board. The first two points do not require any user intervention while the diagnostic can be handled by the user.

There are two kinds of diagnostic:

- general diagnostic (temperatures, power consumption, etc.)
- real time diagnostic (storage of the real time records and upload to the supervisor using a dedicated mechanism called master-BCU see 8.4)

The code and the variables are all mapped in the SRAM memory, the following paragraphs describe the mapping of the relevant variables of the NIOS code that are placed in a fixed shared region of the SRAM.

7.5.1 BCU-NIOS variables

The BCU board shared memory is organized as described in Table 7.

structure	memory location (dword based)	size (dword)
BCU fixed	0x38000	28
BCU system diagnostic	0x3801C	13

Table 7 - BCU NIOS shared memory

The memory is divided into sub-area the *fixed* area contains the fixed configuration parameters of the board (e.g. serial number, IP address, etc.), and the control register that can be modified by the user, as described in Table 8. The second (diagnostic) area contains the diagnostic parameters of the board that are continuously updated by the housekeeping firmware, as described in Table 9. From the general diagnostic structure, for the ARGOS BUS system, only the two temperature sensors are relevant, all the others elements are not available.

The two structures described in Table 8 and Table 9 report all the diagnostic variables available on the BCU shared memory area. For the ARGOS mini-crate system just some of them are relevant as indicated in the third column while all the others should be discarded.

na_bcu_nios_fixed_area_struct		used for ARGOS	brief description
uint16	crateID	NO	crate identification number
uint16	who_ami	NO	bus slot identification number
uint32	software_release	YES	Nios software release Vxx.xx.xxxx
uint16	logic_release	YES	FPGA logic release Vxx.xx
uint8	mac_address[6]	YES	MAC address of Ethernet card
uint8	ip_address[4]	YES	IP address of BCU board
uint8	crate_configuration[20]	YES	array with the bus configuration for each slot
uint16	local_current_threshold	NO	threshold of local power current (without sign)
uint16	vp_set	NO	power voltage set point (positive and negative)
uint16	total_current_thres_pos	NO	threshold of total positive power current
uint16	total_current_thres_neg	NO	threshold of total negative power current
uint32	frames_counter	YES	global counter of the diagnostic frame records stored in SDRAM
uint32	relais_board_out	NO	status of the first relay board (if used)
uint16	serial_number	YES	board serial number
uint16	pb_serial_number	NO	power backplane serial number
uint32	fl_port_inuse	YES	status of the fast link port
uint32	enable_hl_commands	NO	flag to enable(1)/disable(0) the Ethernet hot link commands to the MMT secondary mirror
uint32	enable_pos_acc	NO	flag to enable(1)/disable(0) the reading from MMT secondary mirror of the position accumulation
uint32	diagnostic_record_ptr	YES	pointer to the DSP memory of the frame diagnostic to save in SDRAM
uint32	diagnostic_record_len	YES	size of the diagnostic frame record to save in SDRAM (in DWORDs)
uint16	enable_master_diag	YES	flag to enable(1)/disable(0) the master-BCU mechanism
uint16	decimation_factor	YES	decimation factor of the diagnostic frames to send via master-BCU
uint8	remote_mac_address[6]	YES	MAC address of the remote server to send the diagnostic frames
uint8	remote_ip_address[4]	YES	IP address of the remote server to send the diagnostic frames
uint16	remote_udp_port	YES	UDP port of the remote server to send the diagnostic frames
uint32	rd_diagnostic_record_ptr	YES	current SDRAM pointer of the diagnostic frame to

			send via master-BCU
uint32	wr_diagnostic_record_ptr	YES	current SDRAM pointer to save the diagnostic frame from DSP memory
uint32	rd_byte_serialAIA	NO	number of bytes received from the CCD controller via AIA serial link
uint32	wr_byte_serialAIA	NO	number of bytes to send to the CCD controller via AIA serial link
na_bcu_diagnostic_struct	BCU_diagnostic	YES	area with the BCU system diagnostic, see Table 9

Table 8 – BCU fixed area

na_bcu_diagnostic_struct		**used for ARGOS**	**brief description**
int16	stratix_temp	YES	internal FPGA stratix temperature
int16	power_temp	YES	PCB power area temperature
uint32	bck_digitalIO	NO	status of MAX7301 (U1) digital IO signals on power backplane
uint16	voltage_vccl	NO	voltage level of logic rail monitored by AD7927 (U30) on power backplane
uint16	voltage_vcca	NO	voltage level of positive analogical rail monitored by AD7927 (U30) on power backplane
uint16	voltage_vssa	NO	voltage level of negative analogical rail monitored by AD7927 (U30) on power backplane
uint16	voltage_vccp	NO	voltage level of positive power rail monitored by AD7927 (U30) on power backplane
uint16	voltage_vssp	NO	voltage level of negative power rail monitored by AD7927 (U30) on power backplane
uint16	current_vccl	NO	electrical current of logic rail monitored by AD7927 (U30) on power backplane
uint16	current_vcca	NO	electrical current of positive analogical rail monitored by AD7927 (U30) on power backplane
uint16	current_vssa	NO	electrical current of negative analogical rail monitored by AD7927 (U30) on power backplane
uint16	current_vccp	NO	electrical current of positive power rail monitored by AD7927 (U31) on power backplane
uint16	current_vssp	NO	electrical current of negative power rail monitored by AD7927 (U31) on power backplane
uint16	total_current_vccp	NO	total electrical current of positive power rail monitored by AD7927 (U31) on power backplane
uint16	total_current_vssp	NO	total electrical current of negative power rail monitored by AD7927 (U31) on power backplane
uint16	total_current_vp	NO	total absolute electrical current of power rail monitored by AD7927 (U31) on power backplane
uint16	in0_temp	NO	external temperature sensor #0, monitored by AD7927 (U32) on power backplane
uint16	in1_temp	NO	external temperature sensor #1, monitored by AD7927 (U32) on power backplane
uint16	out0_temp	NO	external temperature sensor #2, monitored by AD7927 (U32) on power backplane
uint16	out1_temp	NO	external temperature sensor #3, monitored by AD7927 (U32) on power backplane
uint16	ext_humidity	NO	external humidity sensor, monitored by AD7927 (U32) on power backplane
uint16	pressure	NO	external coolant pressure sensor, monitored by AD7927 (U32) on power backplane
uint16	dummy_word	NO	not used word
uint32	reset_status	YES	status of BCU reset lines

Table 9 – BCU diagnostic area

7.5.2 DSP-ADC NIOS variables

The DSP-ADC board shared memory is organized as described in Table 10.

structure	memory location (dword based)	size (dword)
DSP fixed	0x18000	505
DSP system diagnostic	0x181F9	36

Table 10 – DSP-ADC NIOS shared memory

Similarly to the BCU shared memory, also the DSP-ADC memory is divided into sub-area the *fixed* area contains the fixed configuration parameters of the board (e.g. serial number, calibration parameters, etc.) and the control register that can be modified by the user; as described in Table 11. The second (diagnostic) area contains the diagnostic parameters of the board that are continuously updated by the housekeeping firmware, as described in Table 12.

The two structures described in Table 11 and Table 12 report all the diagnostic variables available on the DSP shared memory area. For the ARGOS mini-crate system just some of them are relevant as indicated in the third column while all the others should be discarded.

na_dsp_nios_fixed_area_struct		used for ARGOS	brief description
uint8	nios_interpreter_area[369]	NO	shared memory for internal use
uint16	who_ami	NO	bus slot identification number
uint16	logic_release	YES	FPGA logic release Vxx.xx
uint32	software_release	YES	Nios software release Vxx.xx.xxxx
uint16	serial_number	YES	board serial number
uint16	dummy_word	NO	dummy word
uint32	wesp_values	NO	WESP DAC values
float	ADC_spi_curr_offset[16]	NO	calibration values of SPI ADC diagnostic sensors
float	ADC_spi_curr_gain[16]	NO	calibration values of SPI ADC diagnostic sensors
float	ADC_spi_volt_offset[16]	NO	calibration values of SPI ADC diagnostic sensors
float	ADC_spi_volt_gain[16]	NO	calibration values of SPI ADC diagnostic sensors
float	ADC_offset[16]	NO	calibration values of ADC capacitive sensor
float	ADC_gain[16]	NO	calibration values of ADC capacitive sensor
float	DAC_offset[16]	NO	calibration values of DAC coil drivers
float	DAC_gain[16]	NO	calibration values of DAC coil drivers
uint32	diagnostic_record_ptr	YES	pointer to the DSP memory of the frame diagnostic to save in SDRAM
uint32	diagnostic_record_len	YES	size of the diagnostic frame record to save in SDRAM (in DWORDs)
uint32	rd_diagnostic_record_ptr	YES	current SDRAM pointer of the diagnostic frame to send via master-BCU
uint32	wr_diagnostic_record_ptr	YES	current SDRAM pointer to save the diagnostic frame from DSP memory
na_dsp_diagnostic_struct	diagnostic_area	YES	area with the board system diagnostic, see Table 12

Table 11 – DSP-ADC fixed area

na_dsp_diagnostic_struct	used for	brief description

		ARGOS	
int16	stratix_temp	YES	internal FPGA stratix temperature
int16	power_temp	YES	PCB power area temperature
int16	dsps_temp	YES	PCB DSPs area temperature
int16	driverA_temp	YES	PCB driver area temperature (point A)
int16	driverB_temp	YES	PCB driver area temperature (point B)
uint16	dummy_word	NO	dummy word
uint32	driver_status	NO	reset and coil lines status
uint16	coil_current[16]	NO	current of coil drivers
uint16	coil_voltage[16]	NO	voltage of coil drivers

Table 12 – DSP-ADC diagnostic area

7.5.3 HVC-NIOS variables

The HVC board shared memory is organized as described in Table 10.

structure	memory location (dword based)	size (dword)
DSP fixed	0x18000	424
DSP system diagnostic	0x181A8	7

Table 13 - HVC NIOS shared memory

Similarly to the BCU shared memory, also the HVC memory is divided into sub-area the *fixed* area contains the fixed configuration parameters of the board (e.g. serial number, calibration parameters, etc.) and the control register that can be modified by the user; as described in Table 14. The second (diagnostic) area contains the diagnostic parameters of the board that are continuously updated by the housekeeping firmware; as described in Table 15. From the general diagnostic structure, for the ARGOS BUS system, only the four temperature sensors are relevant, all the others elements are not available.

The two structures described in Table 14 and Table 15 report all the diagnostic variables available on the DSP shared memory area. For the ARGOS mini-crate system just some of them are relevant as indicated in the third column while all the others should be discarded.

na_hvc_nios_fixed_area_struct		used for ARGOS	brief description
uint8	nios_interpreter_area[369]	NO	shared memory for internal use
uint16	who_ami	NO	bus slot identification number
uint16	logic_release	YES	FPGA logic release Vxx.xx
uint32	software_release	YES	Nios software release Vxx.xx.xxxx
uint16	serial_number	YES	board serial number
uint16	dummy_word	NO	dummy word
float	ADC_spi_curr_offset[8]	NO	calibration values of SPI ADC diagnostic sensors
float	ADC_spi_curr_gain[8]	NO	calibration values of SPI ADC diagnostic sensors
float	ADC_offset[8]	NO	calibration values of ADC capacitive sensor
float	ADC_gain[8]	NO	calibration values of ADC capacitive sensor

float	DAC_offset[8]	NO	calibration values of DAC coil drivers
float	DAC_gain[8]	NO	calibration values of DAC coil drivers
uint32	diagnostic_record_ptr	YES	pointer to the DSP memory of the frame diagnostic to save in SDRAM
uint32	diagnostic_record_len	YES	size of the diagnostic frame record to save in SDRAM (in DWORDs)
uint32	rd_diagnostic_record_ptr	YES	current SDRAM pointer of the diagnostic frame to send via master-BCU
uint32	wr_diagnostic_record_ptr	YES	current SDRAM pointer to save the diagnostic frame from DSP memory
na_hvc_diagnostic_struct	diagnostic_area	YES	area with the board system diagnostic, see Table 15

Table 14 – HVC fixed area

na_hvc_diagnostic_struct		**used for ARGOS**	**brief description**
int16	stratix_temp	YES	internal FPGA stratix temperature
int16	power_temp	YES	PCB power area temperature
int16	dsps_temp	YES	PCB DSPs area temperature
int16	driver_temp	YES	PCB driver area temperature
uint32	driver_status	NO	reset and coil lines status
uint16	coil_current[8]	NO	current of coil drivers

Table 15 – HVC diagnostic area

8 LGS BCU MINI-CRATE FIRMWARE INITIALIZATION

In this chapter there is the complete description of the firmware variables structure that should be initialized to have the system ready for the real time operations.

8.1 pnCCD pixel interfaces

In order to optimize the real time computation, the pnCCD pixel parameters should be remapped according to the data flux in the algorithm computation, thus avoiding data remapping during the real time code execution, which requires time reducing the computation efficiency.

After introducing the new DSP-ADC board two optional pixel interfaces are available, either the optical fiber interface or the direct analog inputs connected to the on-board ADCs. The two interfaces are designed to download the pixel with the same map; the only difference is related to the header and the footer as described in 8.1.1 and 8.1.2.

For both interfaces the pixel order is the same and to simplify the remapping order, as described in the next paragraphs, we defined a standard pixel numbering for the entire pnCCD pixels frame starting from 0 for the first pixel in the top left corner and increasing the pixel numbering following with highest priority the CCD line order and then the CCD channel order. The Figure 8 shows the pixel numbering order. According to the data format described in the [AD1] the Table 16 shows the pixel correlation.

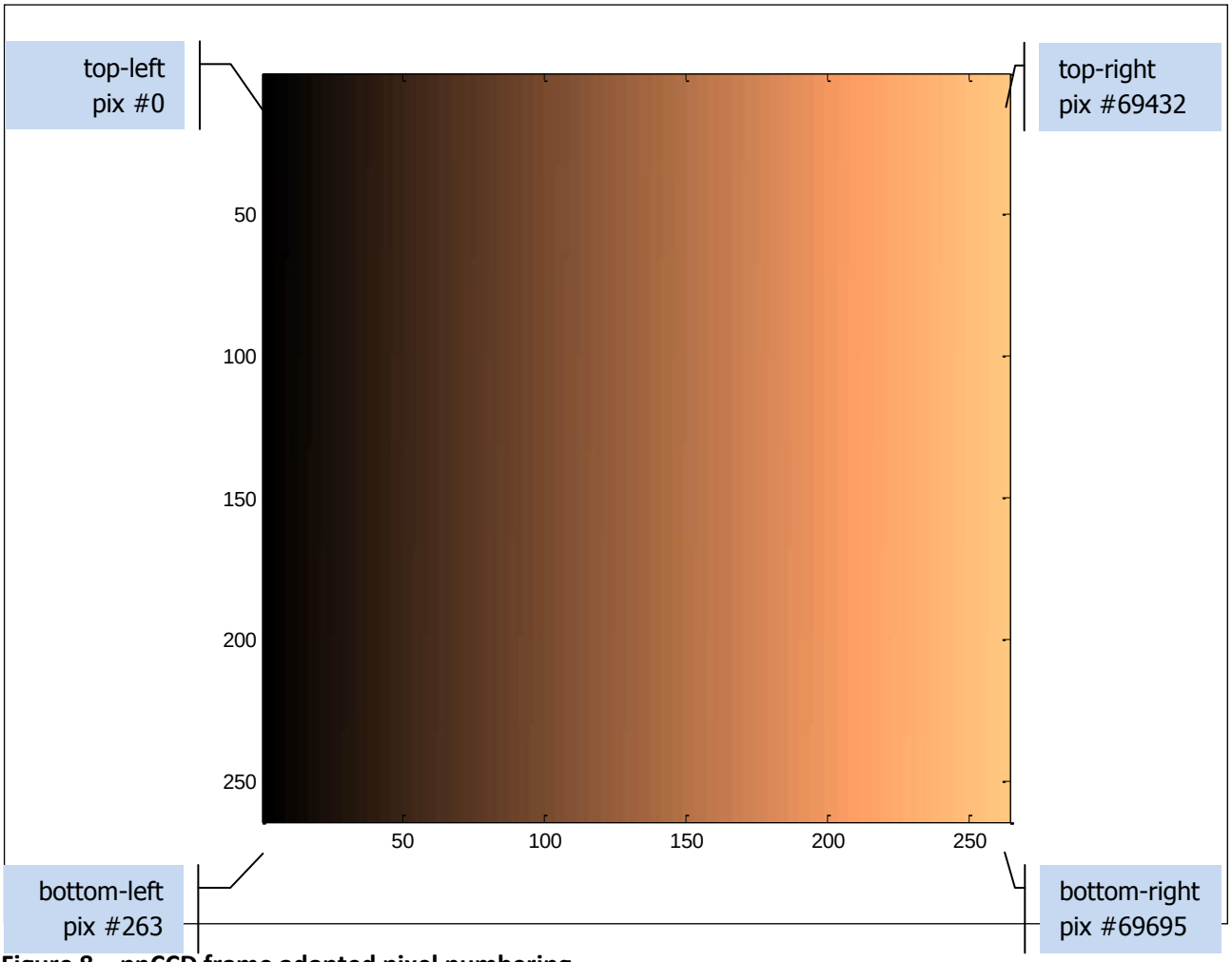

Figure 8 – pnCCD frame adopted pixel numbering

8.1.1 pnCCD pixel map for fiber optic interface

The pnCCD fiber optic interface is implemented in VHDL and it runs in the FPGA of the BCU board. The pnCCD controller is provided with two fiber optic outputs for a parallel download. The protocol is based on standard 0 and 1 Fibre Channel layers ; the higher level is a custom protocol that uses the 8b/10b special chars in order to optimize the data transfer vs. packet overhead, see [AD2]. The BCU FPGA interface protocol has been designed to be compliant with this protocol. The pixels that arrive to the BCU are processed and forwarded to the DSP-ADC for the slope computation according to the DSP firmware.

As above mentioned the optical fiber interface includes two parallel input fibers, from the first interface we received the left half pnCCD pixels (from 0 to 34847 according to the pixels ordering of Figure 8) and from the second one the right half pnCCD pixels (from 34848 to 69695).

Table 16 describes how the frames arrive from optical interface #0, a similar sequence arrives from the second channel.

special word + 3 bytes				First special word of pnCCD fiber link #1	
frame number				Frame number of pnCCD fiber link #1	
time stamp MSW				High part of time stamp of pnCCD fiber link #1	
time stamp LSW				Low part of time stamp of pnCCD fiber link #1	
line 0	pix 66	line 0	pix 0	pix #66	pix #0
line 0	pix 198	line 0	pix 132	pix #198	pix #132
line 0	pix 67	line 0	pix 1	pix #67	pix #1
line 0	pix 199	line 0	pix 133	pix #199	pix #133
line 0	pix 68	line 0	pix 2	pix #68	pix #2
line 0	pix 200	line 0	pix 134	pix #200	pix #134
……		……		……	……
line 0	pix 131	line 0	pix 65	pix #131	pix #65
line 0	pix 263	line 0	pix 197	pix #263	pix #197
line 1	pix 66	line 1	pix 0	pix #330	pix #264
line 1	pix 198	line 1	pix 132	pix #462	pix #396
line 1	pix 67	line 1	pix 1	pix #331	pix #265
line 1	pix 199	line 1	pix 133	pix #463	pix #397
……		……		……	……
……		……		……	……
……		……		……	……
line 131	pix 130	line 131	pix 64	pix #34714	pix #34648
line 131	pix 262	line 131	pix 196	pix #34846	pix #34780
line 131	pix 131	line 131	pix 65	pix #34715	pix #34649
line 131	pix 263	line 131	pix 197	pix #34847	pix #34781

Table 16 – Pixel order arriving from fiber channel #0

For the use of the system is important to clarify some aspect of the pixel download, which is optimized for the slope computation:

- First of all there is a flag to enable/disable the pixel download. In case of data transfer disabled, the pixels arrive to the BCU interface but they are discarded at FPGA level, it means that the DSP doesn't see any pixel and the computation is not executed. Considering that the pnCCD to BCU link protocol doesn't implement any kind of reply/echo the pnCCD is not informed that the pixel frame has not been processed. Another important aspect is that the enable/disable flag is atomic with the pixel start of frame, so no partial pixel download is generated during the interface enabling or disabling. See 8.4.9.2 for the command syntax.
- The pixels are downloaded into a fixed area of the DSP devices of the DSP-ADC board according to the Excel spread sheet, the same frame is downloaded in parallel to both DSP devices of the board. It means that the pointer *_dscub_PixelAreaPtr* variable can't be changed without modifying the BCU Nios firmware. Even if it is possible to modify that pointer, the pixel area has been placed optimizing as much as possible the DSP memory resources so there isn't any reason to modify this pointer apart in case of future modification and/or integration of the slope algorithm.

- The pixels are downloaded at 16bit, the FPGA interface module simply forwards the pixels; conversely the extraction, conversion in floating point precision and then correction of the pixels is completely done by the DSP firmware.
- Just the whole pixel frame is downloadable (is not possible to define CCD sub-areas or remove some lines or channels), Table 17 describes the frame pixel order downloaded in the DSP memory:

DSP memory area	**32 bit dword (uin32 + 2 x uint16)**		
_dscub_PixelArea	special word + 3 bytes		replaced with null when the frame is stored
_dscub_PixelArea + 1	special word + 3 bytes		replaced with frame number when the frame is stored
_dscub_PixelArea + 2	frame number		replaced with time stamp MSW when the frame is stored
_dscub_PixelArea + 3	frame number		replaced with time stamp LSW when the frame is stored
_dscub_PixelArea + 4	time stamp MSW		replaced with internal frame counter (*_dscub_FramesCounter*) when the frame is stored
_dscub_PixelArea + 5	time stamp MSW		replaced with param selector (*_dscub_ParamSelector*) when the frame is stored
_dscub_PixelArea + 6	time stamp LSW		replaced with null when the frame is stored
_dscub_PixelArea + 7	time stamp LSW		replaced with null when the frame is stored
_dscub_PixelArea + 8	pix #66	pix #0	
_dscub_PixelArea + 9	pix #69498	pix #69432	
_dscub_PixelArea + 10	pix #198	pix #132	
_dscub_PixelArea + 11	pix #69630	pix #69564	
_dscub_PixelArea + 12	pix #67	pix #1	
_dscub_PixelArea + 13	pix #69499	pix #69433	
_dscub_PixelArea + 14	pix #199	pix #133	
_dscub_PixelArea + 15	pix #69631	pix #69565	
_dscub_PixelArea + 16	pix #68	pix #2	
_dscub_PixelArea + 17	pix #69500	pix #69434	
_dscub_PixelArea + 18	pix #200	pix #134	
_dscub_PixelArea + 19	pix #69632	pix #69566	
……	……	……	
_dscub_PixelArea + 268	pix #131	pix #65	
_dscub_PixelArea + 269	pix #69563	pix #69497	
_dscub_PixelArea + 270	pix #263	pix #197	
_dscub_PixelArea + 271	pix #69695	pix #69629	
_dscub_PixelArea + 272	pix #330	pix #264	
_dscub_PixelArea + 273	pix #69234	pix #69168	
_dscub_PixelArea + 274	pix #462	pix #396	
_dscub_PixelArea + 275	pix #69366	pix #69300	
_dscub_PixelArea + 276	pix #331	pix #265	
_dscub_PixelArea + 277	pix #69235	pix #69169	
_dscub_PixelArea + 278	pix #463	pix #397	
_dscub_PixelArea + 279	pix #69367	pix #69301	
……	……	……	
……	……	……	
……	……	……	
_dscub_PixelArea + 34848	pix #34714	pix #34648	

_dscub_PixelArea + 34849	pix #34978	pix #34912	
_dscub_PixelArea + 34850	pix #34846	pix #34780	
_dscub_PixelArea + 34851	pix #35110	pix #35044	
_dscub_PixelArea + 34852	pix #34715	pix #34649	
_dscub_PixelArea + 34853	pix #34979	pix #34913	
_dscub_PixelArea + 34854	pix #34847	pix #34781	
_dscub_PixelArea + 34855	pix #35111	pix #35045	

Table 17 – Pixel order on DSP memory area

- all pixels are downloaded to both DSP devices and setting properly the variable *_dscub_HalfCCDSelection*, we command the DSP firmware to use the even or odd pixels for the algorithm. By default the DSP #0 is used to compute the slopes of the left part of the CCD (CAMEX #1 and #2) and the DSP #1 is used to compute the right part of the CCD (CAMEX #3 and #4).

8.1.2 pnCCD pixel map for direct analog acquisition

In case of the new direct acquisition of pnCCD pixels is used the pixel are converted by the 8 ADC devices installed on the DSP-ADC board and the digital values are directly written to the two DSP devices of DSP-ADC board without passing through the BCU board. The data transfer module has been designed to rearrange the incoming pixels to replicate the same pixel sequence of the fiber optic interface. In this way no firmware difference exists at the computational level. The only difference is the starting point of the pixels stream, in fact the header has been reduced from 8 dwords to 4 dwords and at the end of the frame a footer of 4 dwords, that replicates the header, has been added . Note that on this way the total length of the frame is the same (8 + 34848 = 34856).

DSP memory area	**32 bit dword (uin32 + 2 x uint16)**		
_dscub_PixelArea	null		replaced with internal frame counter (*_dscub_FramesCounter*) when the frame is stored
_dscub_PixelArea + 1	null		replaced with param selector (*_dscub_ParamSelector*) when the frame is stored
_dscub_PixelArea + 2	null		null when the frame is stored
_dscub_PixelArea + 3	null		null when the frame is stored
_dscub_PixelArea + 4	pix #66	pix #0	
_dscub_PixelArea + 5	pix #69498	pix #69432	
_dscub_PixelArea + 6	pix #198	pix #132	
_dscub_PixelArea + 7	pix #69630	pix #69564	
_dscub_PixelArea + 8	pix #67	pix #1	
_dscub_PixelArea + 9	pix #69499	pix #69433	
_dscub_PixelArea + 10	pix #199	pix #133	
_dscub_PixelArea + 11	pix #69631	pix #69565	
_dscub_PixelArea + 12	pix #68	pix #2	
_dscub_PixelArea + 13	pix #69500	pix #69434	
_dscub_PixelArea + 14	pix #200	pix #134	

_dscub_PixelArea + 15	pix #69632	pix #69566	
……	……	……	
_dscub_PixelArea + 264	pix #131	pix #65	
_dscub_PixelArea + 265	pix #69563	pix #69497	
_dscub_PixelArea + 266	pix #263	pix #197	
_dscub_PixelArea + 267	pix #69695	pix #69629	
_dscub_PixelArea + 268	pix #330	pix #264	
_dscub_PixelArea + 269	pix #69234	pix #69168	
_dscub_PixelArea + 270	pix #462	pix #396	
_dscub_PixelArea + 271	pix #69366	pix #69300	
_dscub_PixelArea + 272	pix #331	pix #265	
_dscub_PixelArea + 273	pix #69235	pix #69169	
_dscub_PixelArea + 274	pix #463	pix #397	
_dscub_PixelArea + 275	pix #69367	pix #69301	
……	……	……	
……	……	……	
……	……	……	
_dscub_PixelArea + 34844	pix #34714	pix #34648	
_dscub_PixelArea + 34845	pix #34978	pix #34912	
_dscub_PixelArea + 34846	pix #34846	pix #34780	
_dscub_PixelArea + 34847	pix #35110	pix #35044	
_dscub_PixelArea + 34848	pix #34715	pix #34649	
_dscub_PixelArea + 34849	pix #34979	pix #34913	
_dscub_PixelArea + 34850	pix #34847	pix #34781	
_dscub_PixelArea + 34851	pix #35111	pix #35045	
_dscub_PixelArea + 34852	null		replaced with internal frame counter (*_dscub_FramesCounter*) when the frame is stored
_dscub_PixelArea + 34853	null		replaced with param selector (*_dscub_ParamSelector*) when the frame is stored
_dscub_PixelArea + 34854	null		null when the frame is stored
_dscub_PixelArea + 34855	null		null when the frame is stored

Table 18 – Pixel order on DSP memory area

For what concerns the interface enabling and disabling, the pixel download and the DSP memory map location, the firmware mechanisms are the same of the fiber optic interface, refer to 8.1.1 for the description.

8.2 Slope algorithm implementation on DSP-ADC board

The slope algorithm requested in [AD1] is mainly implemented in the two DSP devices of the DSP-ADC board. Each DSP is dedicated for the slope computation of half CCD divided by lines (CAMEX #1 and #2 for DSP #0 and CAMEX #3 and #4 for DSP #1) in order to have almost the same number of centroids to compute for each DSP.

Then the results are passed to the DSP of the BCU board for the slopes reordering, LGS TT slopes and field pointing computation and collection of the slope vector to pass to the DM.

In the DSP devices of DSP-ADC board the computation is mainly divided in pixel correction, centroid computation and slope computation.

The entire computation is reset by the start of frames, which resets the computational machine and prepares the firmware for the new computation.

The pixel correction part is triggered by the number of lines downloaded to the DSP memory. In fact, in order to reduce the computation lag, the entire slope algorithm can be executed in parallel with the pixel download. To do this, the pnCCD interface, after each line download, includes a second special write command to update the *_dscub_NumLinesToDo* variable with the number of lines downloaded, the DSP firmware is in polling to that variable and as soon as it changes, the pixel correction of that line is executed as described in 8.2.1.

As second step the centroid computation can be done just when a certain number of lines are downloaded, to trigger this second part of the algorithm the firmware includes a vector, pointed by the *_dscub_NumSubapsReadyPtr* variable, that should be initialized with the number of slopes that can be computed based on the lines already downloaded and available on memory.

Optimizing as much as possible this mechanism, the computation delay can be really reduced to a minimal delay after the download of the last CCD pixel.

8.2.1 Dark/Background pixel correction

The first pixel correction is the dark/background, simply called offset; this step is done with the pixel still in integer so the pixel offset must be passed in int16 format. The result of the correction is automatically converted to float precision for the rest of the computation.

The pixels offset must be downloaded at the system initialization it the DSP memory area pointed by *_dscub_PixelOffAreaPtr*. The format is the following:

DSP memory area	**32 bit dword (2 x int16)**	
_dscib_PixelOffArea	pix #66	pix #0
_dscib_PixelOffArea + 1	pix #198	pix #132
_dscib_PixelOffArea + 2	pix #67	pix #1
_dscib_PixelOffArea + 3	pix #199	pix #133
_dscib_PixelOffArea + 4	pix #68	pix #2
_dscib_PixelOffArea + 5	pix #200	pix #134
……	……	……
_dscib_PixelOffArea + 130	pix #131	pix #65
_dscib_PixelOffArea + 131	pix #263	pix #197
_dscib_PixelOffArea + 132	pix #330	pix #264
_dscib_PixelOffArea + 133	pix #462	pix #396
_dscib_PixelOffArea + 134	pix #331	pix #265
_dscib_PixelOffArea + 135	pix #463	pix #397
……	……	……
……	……	……
……	……	……

_dscib_PixelOffArea + 17420	pix #34714	pix #34648
_dscib_PixelOffArea + 17421	pix #34846	pix #34780
_dscib_PixelOffArea + 17422	pix #34715	pix #34649
_dscib_PixelOffArea + 17423	pix #34847	pix #34781

Table 19 – Pixel offset order for DSP#0 (CAMEX #1 and #2)

DSP memory area	**32 bit dword (2 x int16)**	
_dscib_PixelOffArea	pix #69498	pix #69432
_dscib_PixelOffArea + 1	pix #69630	pix #69564
_dscib_PixelOffArea + 2	pix #69499	pix #69433
_dscib_PixelOffArea + 3	pix #69631	pix #69565
_dscib_PixelOffArea + 4	pix #69500	pix #69434
_dscib_PixelOffArea + 5	pix #69632	pix #69566
……	……	……
_dscib_PixelOffArea + 130	pix #69563	pix #69497
_dscib_PixelOffArea + 131	pix #69695	pix #69629
_dscib_PixelOffArea + 132	pix #69234	pix #69168
_dscib_PixelOffArea + 133	pix #69366	pix #69300
_dscib_PixelOffArea + 134	pix #69235	pix #69169
_dscib_PixelOffArea + 135	pix #69367	pix #69301
……	……	……
……	……	……
……	……	……
_dscib_PixelOffArea + 17420	pix #34978	pix #34912
_dscib_PixelOffArea + 17421	pix #35110	pix #35044
_dscib_PixelOffArea + 17422	pix #34979	pix #34913
_dscib_PixelOffArea + 17423	pix #35111	pix #35045

Table 20 – Pixel offset order for DSP#1 (CAMEX #3 and #4)

In parallel to the pixel offset correction and conversion the DSP firmware executes also the common mode threshold check, the check is done when the pixel is still in integer format and for this reason the variable *_dscib_CMThreshold* should be configured in integer 32bit format.

8.2.2 Common mode pixel correction

The common mode (CM) correction consists of a line-by-line correction for all pixels (as indicated in the desiderata options of [AD1], we extended the common mode and gain correction to all pixels in the frame). The common mode coefficients are computed as average of the common mode pixels for each line; so to define the CM pixels we introduced a common mode map where for each pixel it must be indicated if it is a CM pixels or not. The CM pixel map has the following format, where for each pixel the corresponding bit must be set to 0 if it is not a CM pixel or set to 1 if it is. Notice that just the 4 least significant bits of each 32bit dword are used.

DSP memory area	32 bit dword (bit 3 - bit 2 - bit 1 - bit 0)			
_dscub_CMPixelMap	pix #198	pix #132	pix #66	pix #0
_dscub_CMPixelMap + 1	pix #199	pix #133	pix #67	pix #1
_dscub_CMPixelMap + 2	pix #200	pix #134	pix #68	pix #2
……	……			
_dscub_CMPixelMap + 64	pix #263	pix #197	pix #131	pix #65
_dscub_CMPixelMap + 65	pix #462	pix #396	pix #330	pix #264
_dscub_CMPixelMap + 66	pix #463	pix #397	pix #331	pix #265
……	……			
_dscub_CMPixelMap + 8710	pix #34846	pix #34780	pix #34714	pix #34648
_dscub_CMPixelMap + 8711	pix #34847	pix #34781	pix #34715	pix #34649

Table 21 – CM pixel map for DSP#0 (CAMEX #1 and #2)

DSP memory area	32 bit dword			
_dscub_CMPixelMap	pix #69630	pix #69564	pix #69498	pix #69432
_dscub_CMPixelMap + 1	pix #69631	pix #69565	pix #69499	pix #69433
_dscub_CMPixelMap + 2	pix #69632	pix #69566	pix #69500	pix #69434
……	……			
_dscub_CMPixelMap + 64	pix #69695	pix #69629	pix #69563	pix #69497
_dscub_CMPixelMap + 65	pix #69366	pix #69300	pix #69234	pix #69168
_dscub_CMPixelMap + 66	pix #69367	pix #69301	pix #69235	pix #69169
……	……			
_dscub_CMPixelMap + 8710	pix #35110	pix #35044	pix #34978	pix #34912
_dscub_CMPixelMap + 8711	pix #35111	pix #35045	pix #34979	pix #34913

Table 22 – CM pixel map for DSP#1 (CAMEX #3 and #4)

8.2.3 Flat-field/gain correction

After the CM correction the next step is the gain correction which is applied to all pixel frame, as already mentioned in 8.2.2. Respect to the gain reported to [AD1] the value to set in the DSP memory is the reciprocal so that the real time processor executes just a product instead of a division. The gain vector must be set in the DSP memory area pointed by *_dscub_PixelGainAreaPtr* in single precision float value.

Here the remapped order, for each DSP, of the gain vector:

DSP memory area	32 bit dword (float)
_dscfb_PixelGainArea	pix #0
_dscfb_PixelGainArea + 1	pix #66
_dscfb_PixelGainArea + 2	pix #132
_dscfb_PixelGainArea + 3	pix #198
_dscfb_PixelGainArea + 4	pix #1
_dscfb_PixelGainArea + 5	pix #67

_dscfb_PixelGainArea + 6	pix #133
_dscfb_PixelGainArea + 7	pix #199
_dscfb_PixelGainArea + 8	pix #2
_dscfb_PixelGainArea + 9	pix #68
_dscfb_PixelGainArea + 10	pix #134
_dscfb_PixelGainArea + 11	pix #200
……	……
_dscfb_PixelGainArea + 260	pix #65
_dscfb_PixelGainArea + 261	pix #131
_dscfb_PixelGainArea + 262	pix #197
_dscfb_PixelGainArea + 263	pix #263
_dscfb_PixelGainArea + 264	pix #264
_dscfb_PixelGainArea + 265	pix #330
_dscfb_PixelGainArea + 266	pix #396
_dscfb_PixelGainArea + 267	pix #462
_dscfb_PixelGainArea + 268	pix #265
_dscfb_PixelGainArea + 269	pix #331
_dscfb_PixelGainArea + 270	pix #397
_dscfb_PixelGainArea + 271	pix #463
……	……
……	……
……	……
_dscfb_PixelGainArea + 34840	pix #34648
_dscfb_PixelGainArea + 34841	pix #34714
_dscfb_PixelGainArea + 34842	pix #34780
_dscfb_PixelGainArea + 34843	pix #34846
_dscfb_PixelGainArea + 34844	pix #34649
_dscfb_PixelGainArea + 34845	pix #34715
_dscfb_PixelGainArea + 34846	pix #34781
_dscfb_PixelGainArea + 34847	pix #34847

Table 23 – Pixel gain order for DSP#0 (CAMEX #1 and #2)

DSP memory area	32 bit dword (float)
_dscfb_PixelGainArea	pix #69432
_dscfb_PixelGainArea + 1	pix #69498
_dscfb_PixelGainArea + 2	pix #69564
_dscfb_PixelGainArea + 3	pix #69630
_dscfb_PixelGainArea + 4	pix #69433
_dscfb_PixelGainArea + 5	pix #69499
_dscfb_PixelGainArea + 6	pix #69565
_dscfb_PixelGainArea + 7	pix #69631
_dscfb_PixelGainArea + 8	pix #69434
_dscfb_PixelGainArea + 9	pix #69500
_dscfb_PixelGainArea + 10	pix #69566
_dscfb_PixelGainArea + 11	pix #69632
……	……
_dscfb_PixelGainArea + 260	pix #69497

_dscfb_PixelGainArea + 261	pix #69563
_dscfb_PixelGainArea + 262	pix #69629
_dscfb_PixelGainArea + 263	pix #69695
_dscfb_PixelGainArea + 264	pix #69168
_dscfb_PixelGainArea + 265	pix #69234
_dscfb_PixelGainArea + 266	pix #69300
_dscfb_PixelGainArea + 267	pix #69366
_dscfb_PixelGainArea + 268	pix #69169
_dscfb_PixelGainArea + 269	pix #69235
_dscfb_PixelGainArea + 270	pix #69301
_dscfb_PixelGainArea + 271	pix #69367
……	……
……	……
……	……
_dscfb_PixelGainArea + 34840	pix #34912
_dscfb_PixelGainArea + 34841	pix #34978
_dscfb_PixelGainArea + 34842	pix #35044
_dscfb_PixelGainArea + 34843	pix #35110
_dscfb_PixelGainArea + 34844	pix #34913
_dscfb_PixelGainArea + 34845	pix #34979
_dscfb_PixelGainArea + 34846	pix #35045
_dscfb_PixelGainArea + 34847	pix #35111

Table 24 – Pixel gain order for DSP#1 (CAMEX #3 and #4)

8.2.4 Centroid computation

The previous computational steps are executed for each pixels while, from now, the computation involves just the slope pixels, used for the centroid computation first and slope computation after.

The algorithm is structured to compute centroid of sub-apertures of fixed size of 8x8 pixels. To consider smaller centroid in the computation is possible to set to zero the weighting coefficients (see 8.2.4.4) for all the not used pixels. As already mentioned, in order to optimize the computational time on both devices, is very important to balance the number of sub-apertures on each DSP.

Staring from the optical layout of the three LGS in the CCD area, the DSP code has been implemented and optimized with the following constrains:

- each DSP manages two CAMEX;
- one entire sub-aperture MUST stay in the same DSP;
- as result we have that the CAMEX border MUST match the sub-apertures borders, for all the sub-apertures of the central LGS, see Figure 10;
- the number of sub-apertures for each DSP should be even because the computation is done in parallel with two sub-apertures at a time, the dummy sub-aperture can be the last repeated twice;

- moreover the number of sub-apertures should be incremented by two because the pixel maximum value and centroid computations are done in parallel. Additionally, at the beginning of the process the first centroid is computed twice, the first time for the maximum research and the second time for the centroid computation. Also this is done to optimize the DSP computation.

In Figure 9 there is an example of three LGS with 176 sub-apertures for each pupil. In the example the total number of sub-apertures is 528 and the DSP#0 has to calculate 257 sub-apertures while the DSP #1 has to compute 271 sub-apertures and, considering the previous constrains on number of sub-apertures, the final number is 260 and 274 respectively.

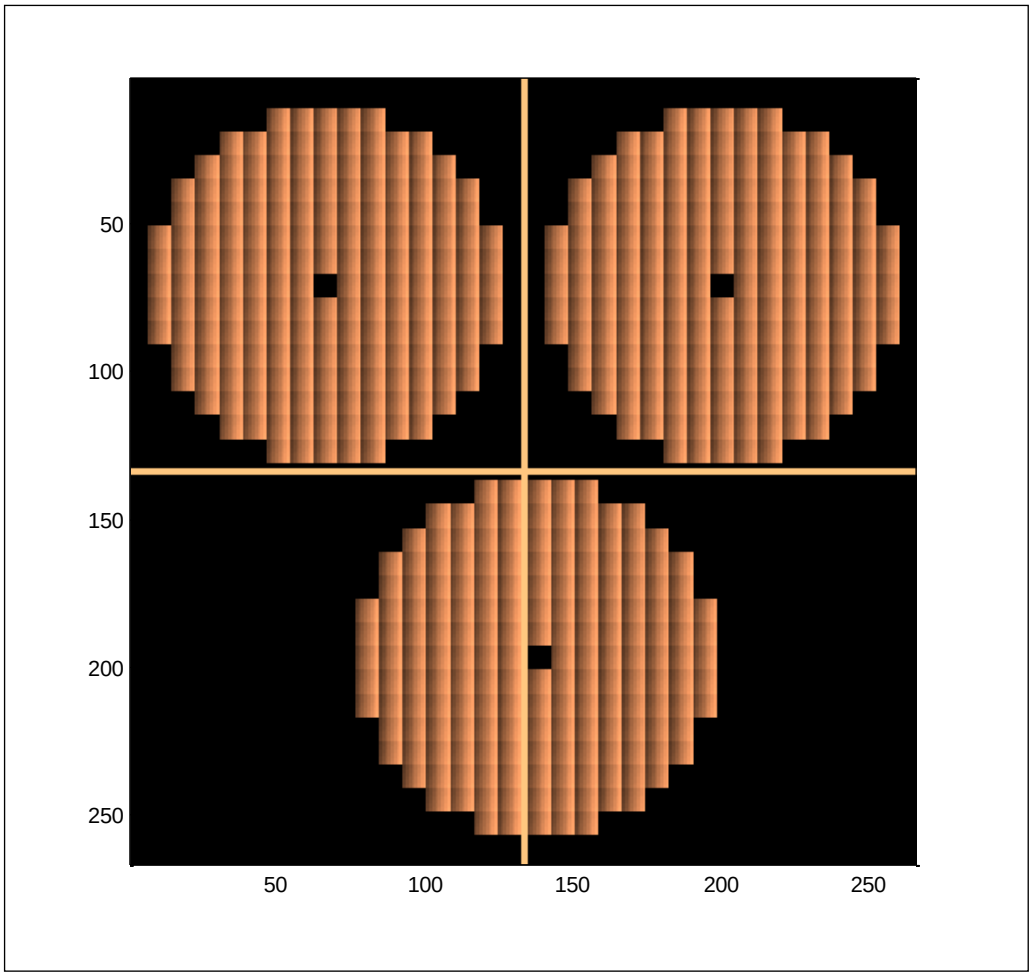

Figure 9 – example of LGS placin in pnCCD frame

And Figure 10 shows the detail of the sub-apertures placing respect with CAMEX #2 and #3 borders.

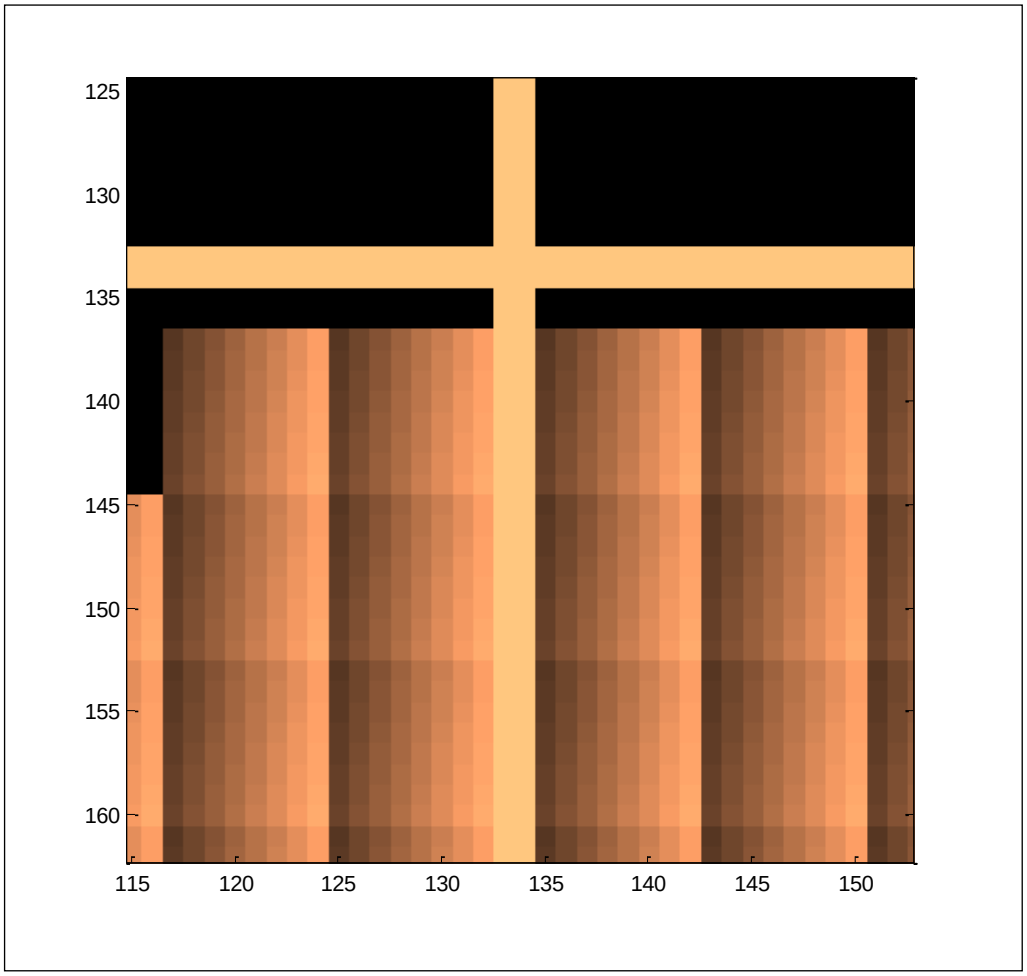

Figure 10 – detail of the sub-apertures placing vs CAMEX borders

8.2.4.1 Definition of sub-apertures pixels

First of all we have to define the slope pixels and the sub-apertures related to them. To do this we have to initialize the vector *_dscub_SubapPixelPtrA* and *_dscub_SubapPixelPtrB*, pointed respectively by *_dscub_SubapPixelPtrAPtr* and *_dscub_SubapPixelPtrBPtr*. These two vectors should be set equal (they are duplicated for the internal optimization mechanism); for each slope pixel, the vectors contain the pointer to the *_dscub_PixelArea* memory area where the slope pixel is located.

Considering the fixed sub-aperture size of 64 pixels, the dimension of *_dscub_SubapPixelPtrA* and *_dscub_SubapPixelPtrB* vectors is 64 x (number of sub-apertures) for each DSP; the definition of the number of sub-apertures, number of slope pixels, number of centroids for DSP etc. should fulfill the constrains indicated in 8.2.4. The Excel spread-sheet [RD1] helps the user to initialize properly the DSP registers.

The sub-aperture numbering should follow for each DSP the computational priority according to the pixel download from the pnCCD. This sequence defines also the centroid and slope output order from each DSP.

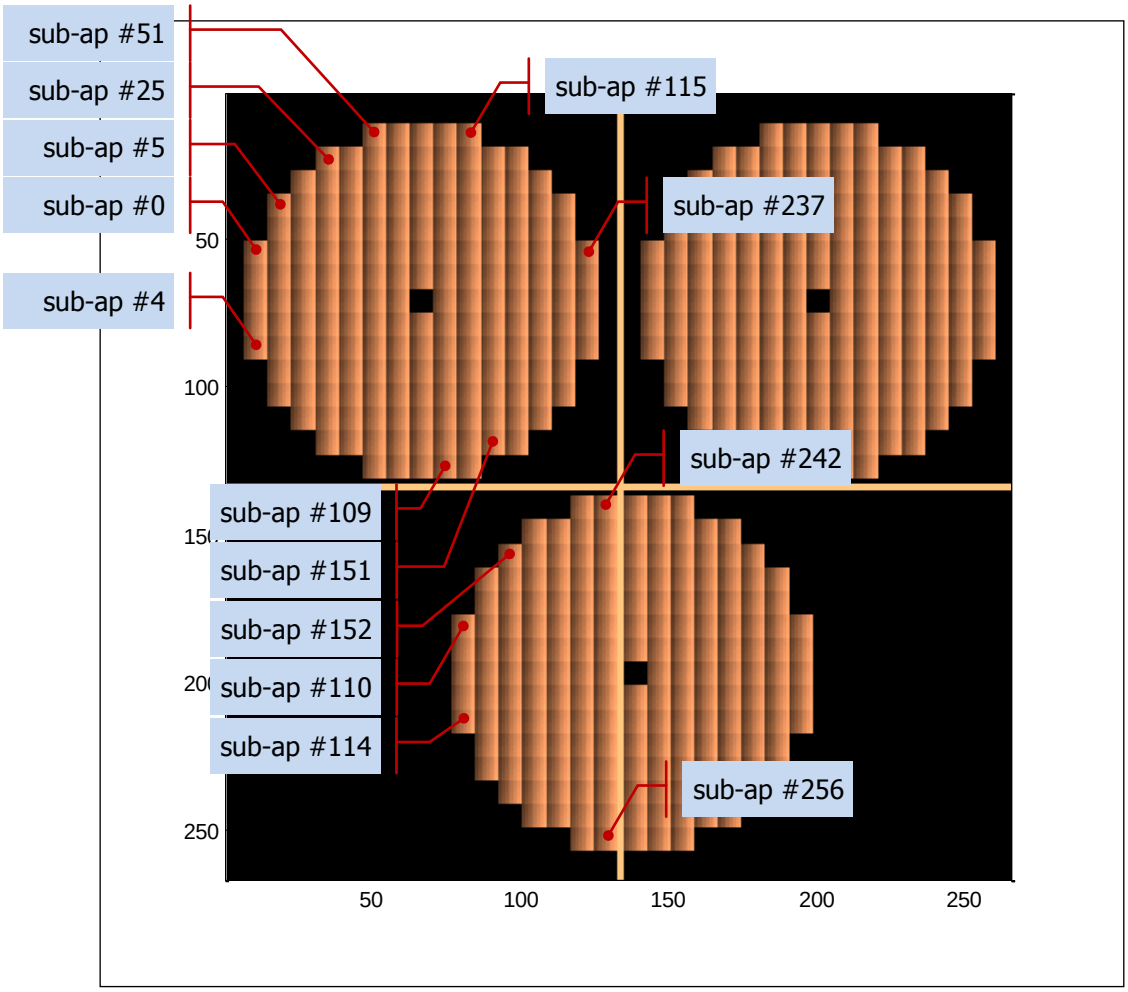

Figure 11 – sub-aperture numbering for DSP #0 (CAMEX #1 and #2)

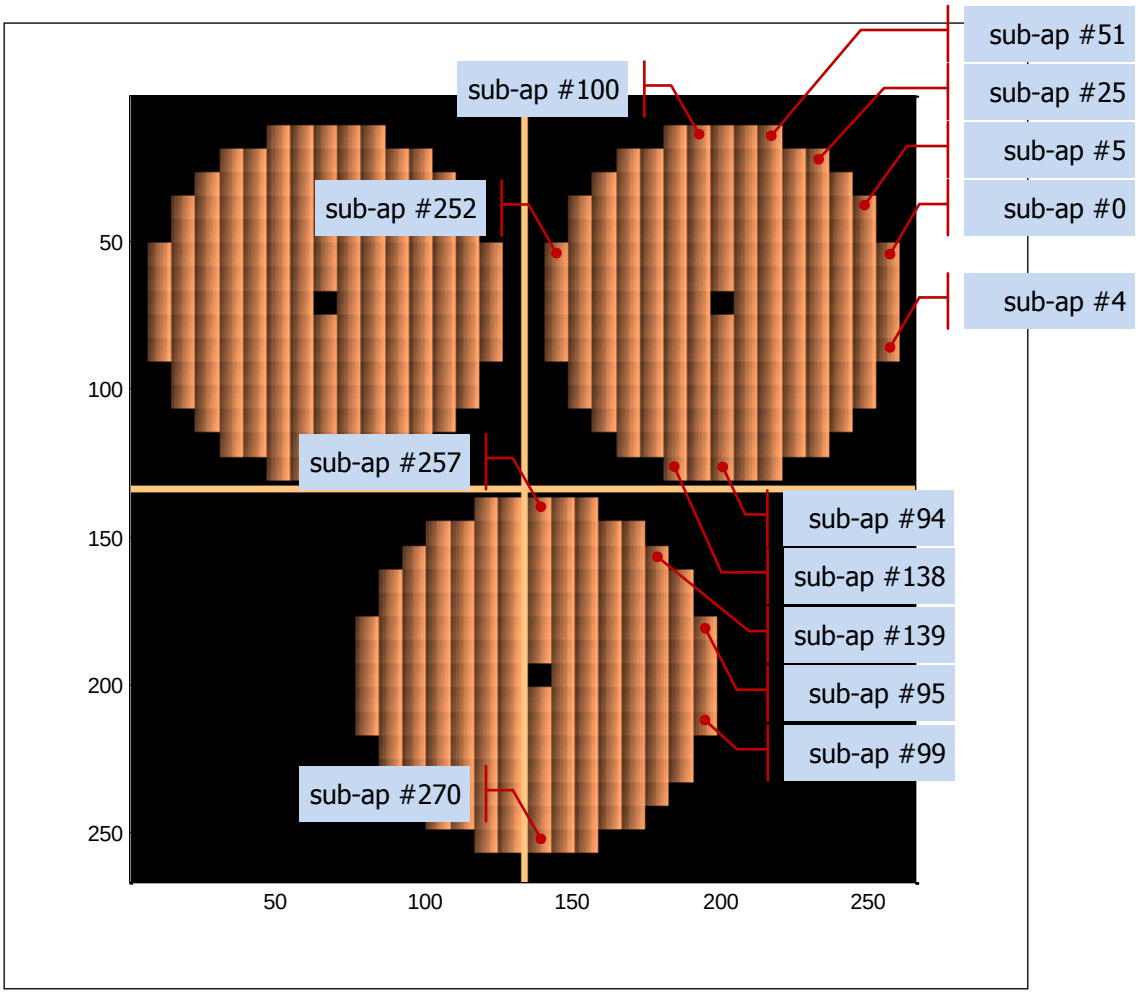

Figure 12 – sub-aperture numbering for DSP #1 (CAMEX #3 and #4)

Similar to the sub-aperture numbering also the sub-aperture pixel numbering should follow the pixel priority download. The Figure 13 shows the pixel order inside the sub-aperture.

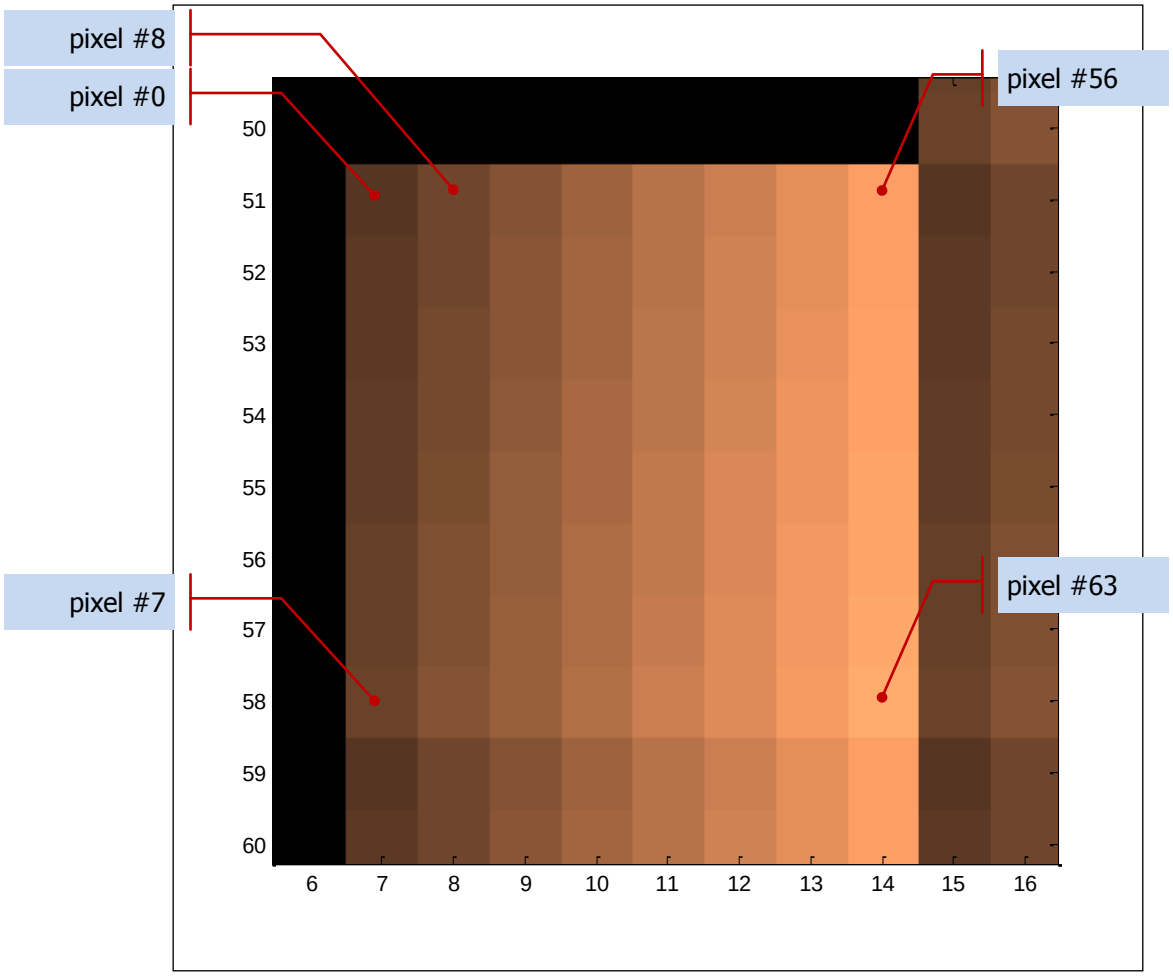

Figure 13 – sub-aperture pixel numbering

Using the previous pictures Table 25 and Table 26 show the organization of the *_dscub_SubapPixelPtrA* and *_dscub_SubapPixelPtrB* vector.

DSP memory area	sub-aperture #	pixel #	32 bit dword (uint32)
_dscub_SubapPixelPtr	0	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 64	1		
_dscub_SubapPixelPtr + 128	0	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 192	1	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 256	2	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 320	3	0 to 63	64 pointers to _dscub_PixelArea
……	……	……	……
_dscub_SubapPixelPtr + 16448	255	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 16512	256	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 16576	256	0 to 63	64 pointers to _dscub_PixelArea

Table 25 – Slope pixel pointer map DSP#0 (CAMEX #1 and #2)

DSP memory area	sub-aperture #	pixel #	32 bit dword (uint32)
_dscub_SubapPixelPtr	0	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 64	1		
_dscub_SubapPixelPtr + 128	0	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 192	1	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 256	2	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 320	3	0 to 63	64 pointers to _dscub_PixelArea
……	……	……	……
_dscub_SubapPixelPtr + 17344	269	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 17408	270	0 to 63	64 pointers to _dscub_PixelArea
_dscub_SubapPixelPtr + 17472	270	0 to 63	64 pointers to _dscub_PixelArea

Table 26 – Slope pixel pointer map DSP#1 (CAMEX #3 and #4)

Note that, based on the consideration above, the first two sub-apertures (#0 and #1) should be repeated twice for internal computational optimization and for the same reason the number of computed slopes should be even and then at the end of the vector we repeated the centroid computation of the last sub-aperture.

8.2.4.2 Centroid computation vs pixels download

Similar to the pixel correction, also the centroid & slope computation can be done in parallel with the pixels download. The computation shall proceed following the pixel downloading process, to do this the *_dscub_NumSubapsReady* vector (defined by *_dscub_NumSubapsReadyPtr* pointer) should be initialized properly.

For each DSP, the vector size is equal to the half number of lines of the pnCCD and should be filled setting for each line downloaded how many slopes the firmware can compute. The Table 27 describes the vector initialization for both DSP devices. Note: initializing the vector, remember that a centroid can be computed only when all the centroid pixels are downloaded.

DSP memory area	32 bit dword (uint32)
_dscub_NumSubapsReady	(number of centroid to compute) / 2 after 1 line downloaded
_dscub_NumSubapsReady + 1	(number of centroid to compute) / 2 after 2 lines downloaded
_dscub_NumSubapsReady + 2	(number of centroid to compute) / 2 after 3 lines downloaded
……	……
_dscub_NumSubapsReady + 131	(number of centroid to compute) / 2 after 132 lines downloaded

Table 27 – Centroid computation trigger vector

8.2.4.3 Centroid x & y arms coefficients

The centroid "arms" are the two vectors for the x and y centroid computation from the sub-aperture pixels. The size of this vector depends on the sub-aperture size, fixed in the firmware at 8x8 = 64 pixels, then the size of the *_dscfb_SubapPixelArms* vector (pointed by *_dscub_SubapPixelArmsPtr* pointer) is 128. The Table 28 shows the vector structure using the same sub-aperture pixel numbering adopted in Figure 13.

DSP memory area	32 bit dword (float)
_dscfb_SubapPixelArms	pixel #0 – x
_dscfb_SubapPixelArms + 1	pixel #0 – y
_dscfb_SubapPixelArms + 2	pixel #1 – x
_dscfb_SubapPixelArms + 3	pixel #1 – y
_dscfb_SubapPixelArms + 4	pixel #2 – x
_dscfb_SubapPixelArms + 5	pixel #2 – y
……	……
_dscfb_SubapPixelArms + 124	pixel #62 – x
_dscfb_SubapPixelArms + 125	pixel #62 – y
_dscfb_SubapPixelArms + 126	pixel #63 – x
_dscfb_SubapPixelArms + 127	pixel #63 – y

Table 28 – Centroid x & y arms coefficients

8.2.4.4 Pixel and sub-aperture weighting function

This coefficient (called $W_{i;a}$ in [AD1]) is a slope pixel and sub-aperture dependant gain coefficient, so the total size of the vector *_dscfb_SubapPixelWeight* (pointed by *_dscub_SubapPixelWeightPtr* pointer) is equal to the total number of sub-apertures (including the dummy ones) times the number of sub-aperture pixels.

As already mentioned this coefficient can be used to set some sub-aperture pixels to zero in case of smaller sub-aperture size.

The Table 29 and Table 30 show the vector structure.

DSP memory area	32 bit dword (float)
_dscfb_SubapPixelWeight	sub-ap #0 - pixel #0
_dscfb_SubapPixelWeight + 1	sub-ap #1 - pixel #0
_dscfb_SubapPixelWeight + 2	sub-ap #0 - pixel #1
_dscfb_SubapPixelWeight + 3	sub-ap #1 - pixel #1
_dscfb_SubapPixelWeight + 4	sub-ap #0 - pixel #2
_dscfb_SubapPixelWeight + 5	sub-ap #1 - pixel #2
……	……
_dscfb_SubapPixelWeight + 126	sub-ap #0 - pixel #63
_dscfb_SubapPixelWeight + 127	sub-ap #1 - pixel #63
_dscfb_SubapPixelWeight + 128	sub-ap #2 - pixel #0

_dscfb_SubapPixelWeight + 129	sub-ap #3 - pixel #0
……	……
_dscfb_SubapPixelWeight + 16636	sub-ap #258 - pixel #62
_dscfb_SubapPixelWeight + 16637	sub-ap #259 - pixel #62
_dscfb_SubapPixelWeight + 16638	sub-ap #258 - pixel #63
_dscfb_SubapPixelWeight + 16639	sub-ap #259 - pixel #63

Table 29 – Weighting function coefficients for DSP#0

DSP memory area	32 bit dword (float)
_dscfb_SubapPixelWeight	sub-ap #0 - pixel #0
_dscfb_SubapPixelWeight + 1	sub-ap #1 - pixel #0
_dscfb_SubapPixelWeight + 2	sub-ap #0 - pixel #1
_dscfb_SubapPixelWeight + 3	sub-ap #1 - pixel #1
_dscfb_SubapPixelWeight + 4	sub-ap #0 - pixel #2
_dscfb_SubapPixelWeight + 5	sub-ap #1 - pixel #2
……	……
_dscfb_SubapPixelWeight + 126	sub-ap #0 - pixel #63
_dscfb_SubapPixelWeight + 127	sub-ap #1 - pixel #63
_dscfb_SubapPixelWeight + 128	sub-ap #2 - pixel #0
_dscfb_SubapPixelWeight + 129	sub-ap #3 - pixel #0
……	……
_dscfb_SubapPixelWeight + 17532	sub-ap #272 - pixel #62
_dscfb_SubapPixelWeight + 17533	sub-ap #273 - pixel #62
_dscfb_SubapPixelWeight + 17534	sub-ap #272 - pixel #63
_dscfb_SubapPixelWeight + 17535	sub-ap #273 - pixel #63

Table 30 – Weighting function coefficients for DSP#1

8.2.4.5 Centroid pixel threshold coefficients

According to [AD1], the centroid pixel threshold could be, for each sub-aperture, either a constant value or determined dynamically as a value being proportional to the maximum sub-aperture pixel. To avoid the selection in the real-time computation, the proposed option has been implemented merging the constant and the dynamical part together, then, using the same nomenclature, the final threshold value is: $\widetilde{I}_{a;T} = I_{a;T} + \alpha_a \cdot I_{a;\max}$ where $I_{a;T}$ is the constant sub-aperture dependent threshold coefficient while $\alpha_a \cdot I_{a;\max}$ is the dynamical sub-aperture dependent threshold coefficient. For the final centroid computation, by setting either the constant threshold or the dynamical gain α_a to zero is possible to disable the first or the second contribution.

Note: concerning the requirements in [AD1], the dynamic gain that is a constant for all sub-apertures, the implemented algorithm provides a sub-aperture dependent dynamic gain, to allow the maximum flexibility in the algorithm implementation.

The two coefficient vectors, dynamic gain and constant threshold respectively, must be saved in *_dscfb_SubapMaxGain* (pointed by *_dscub_SubapMaxGainPtr* pointer) and *_dscfb_SubapFixThreshold* (pointed by *_dscub_SubapFixThresholdPtr* pointer) and organized as shown in Table 31 and Table 32.

DSP memory area		32 bit dword (float)
_dscfb_SubapMaxGain	_dscfb_SubapFixThreshold	sub-ap #0
_dscfb_SubapMaxGain + 1	_dscfb_SubapFixThreshold + 1	sub-ap #1
_dscfb_SubapMaxGain + 2	_dscfb_SubapFixThreshold + 2	sub-ap #2
_dscfb_SubapMaxGain + 3	_dscfb_SubapFixThreshold + 3	sub-ap #3
_dscfb_SubapMaxGain + 4	_dscfb_SubapFixThreshold + 4	sub-ap #4
_dscfb_SubapMaxGain + 5	_dscfb_SubapFixThreshold + 5	sub-ap #5
……	……	……
_dscfb_SubapMaxGain + 258	_dscfb_SubapFixThreshold + 258	sub-ap #258
_dscfb_SubapMaxGain + 259	_dscfb_SubapFixThreshold + 259	sub-ap #259

Table 31 – Centroid threshold coefficients for DSP#0

DSP memory area		32 bit dword (float)
_dscfb_SubapMaxGain	_dscfb_SubapFixThreshold	sub-ap #0
_dscfb_SubapMaxGain + 1	_dscfb_SubapFixThreshold + 1	sub-ap #1
_dscfb_SubapMaxGain + 2	_dscfb_SubapFixThreshold + 2	sub-ap #2
_dscfb_SubapMaxGain + 3	_dscfb_SubapFixThreshold + 3	sub-ap #3
_dscfb_SubapMaxGain + 4	_dscfb_SubapFixThreshold + 4	sub-ap #4
_dscfb_SubapMaxGain + 5	_dscfb_SubapFixThreshold + 5	sub-ap #5
……	……	……
_dscfb_SubapMaxGain + 272	_dscfb_SubapFixThreshold + 272	sub-ap #272
_dscfb_SubapMaxGain + 273	_dscfb_SubapFixThreshold + 273	sub-ap #273

Table 32 – Centroid threshold coefficients for DSP#1

8.2.4.6 Centroid pixel factor *n*

The centroid computation algorithm the centroid pixels minus the centroid threshold should be increased for a power factor *n*. The selection between 1 and 1.5 of the factor is done setting properly the variable *_dscub_SubapPowerCoeff*: setting 0 the *n* factor is set to 1 and setting 1 the factor *n* is set to 1.5.

8.2.4.7 Centroid total flux accumulation

The algorithm includes two optional centroid computations with the respect to the total flux accumulation, the usual centroid computation uses for each sub-aperture the total flux of the sub-aperture itself and this method is used by the algorithm when the variable *_dscub_CentroidAverage* is set to zero (default value).

Setting the variable *_dscub_CentroidAverage* to one the alternative centroid computation is enabled. In this case all centroids of each LGS pupil can be computed using for each LGS pupil the whole LGS pixel flux, or better the sum of all sub-aperture pixels of that LGS pupil. To simplify the firmware architecture at the moment this mechanism is limited to the pupils that fully belong to each DSP, and for the third LGS pupil that it is split between the two DSPs (see Figure 9), the algorithm is implemented considering for each DSP just the sub-apertures that belong to it.

When this second mechanism is used, the vector *_dscub_LGSCentroidMap* (pointed by *_dscub_LGSCentroidMapPtr*) should be initialized to assign at every sub-aperture a LGS pupil. This is done setting each location to 0 if the flux sub-aperture should be skipped, to 1 to add the sub-aperture flux to the LGS pupil #0 accumulation, to 2 for the LGS pupil #1 accumulation and finally to 3 for the LGS pupil #2 accumulation. Values out of the [0,3] range are not accepted.

Table 33 and Table 34 show how the vector should be initialized.

DSP memory area	32 bit dword (uint32)
_dscub_LGSCentroidMap	sub-ap #0
_dscub_LGSCentroidMap + 1	sub-ap #1
_dscub_LGSCentroidMap + 2	sub-ap #2
_dscub_LGSCentroidMap + 3	sub-ap #3
_dscub_LGSCentroidMap + 4	sub-ap #4
_dscub_LGSCentroidMap + 5	sub-ap #5
……	……
_dscub_LGSCentroidMap + 258	sub-ap #258
_dscub_LGSCentroidMap + 259	sub-ap #259

Table 33 – Sub-aperture to LGS pupil assignement for DSP#0

DSP memory area	32 bit dword (uint32)
_dscub_LGSCentroidMap	sub-ap #0
_dscub_LGSCentroidMap + 1	sub-ap #1
_dscub_LGSCentroidMap + 2	sub-ap #2
_dscub_LGSCentroidMap + 3	sub-ap #3
_dscub_LGSCentroidMap + 4	sub-ap #4
_dscub_LGSCentroidMap + 5	sub-ap #5
……	……
_dscub_LGSCentroidMap + 272	sub-ap #272
_dscub_LGSCentroidMap + 273	sub-ap #273

Table 34 – Sub-aperture to LGS pupil assignement for DSP#1

For the proper use of this mechanism also the variables *_dscfb_NumSubapLGS0, _dscfb_NumSubapLGS0* and *_dscfb_NumSubapLGS0* should be initialized respectively with the number of sub-apertures of the LGS pupil #0, #1 and #2 of each DSP.

8.2.4.8 Centroid linear coefficients

To complete the centroid computation, an additional centroid linearization coefficient (called γ_a in [AD1]) is applied to the centroid.

The coefficient vector is saved in *_dscfb_SubapLinearCoeff* (pointed by *_dscub_SubapLinearCoeffPtr* pointer) and organized as shown in Table 35 and Table 36.

DSP memory area	32 bit dword (float)
_dscfb_SubapLinearCoeff	sub-ap #0
_dscfb_SubapLinearCoeff + 1	sub-ap #1
_dscfb_SubapLinearCoeff + 2	sub-ap #2
_dscfb_SubapLinearCoeff + 3	sub-ap #3
_dscfb_SubapLinearCoeff + 4	sub-ap #4
_dscfb_SubapLinearCoeff + 5	sub-ap #5
……	……
_dscfb_SubapLinearCoeff + 258	sub-ap #258
_dscfb_SubapLinearCoeff + 259	sub-ap #259

Table 35 – Centroid linear coefficients for DSP#0

DSP memory area	32 bit dword (float)
_dscfb_SubapLinearCoeff	sub-ap #0
_dscfb_SubapLinearCoeff + 1	sub-ap #1
_dscfb_SubapLinearCoeff + 2	sub-ap #2
_dscfb_SubapLinearCoeff + 3	sub-ap #3
_dscfb_SubapLinearCoeff + 4	sub-ap #4
_dscfb_SubapLinearCoeff + 5	sub-ap #5
……	……
_dscfb_SubapLinearCoeff + 272	sub-ap #272
_dscfb_SubapLinearCoeff + 273	sub-ap #273

Table 36 – Centroid linear coefficients for DSP#1

8.2.5 Slope offset coefficients

The entire computation concludes with the slope computation, according to [AD1] it is obtained subtracting an offset (called $x_{0,a}$ and $y_{0,a}$ in [AD1]) to the computed centroid. As requested, the slope offset can be modified on-the-fly during the real time computation and to do this in atomic way we introduced two sets (banks) of offset coefficients. The selection of the bank to use is triggered by the *_dscub_ParamSelector* variable at bit #0 (see Table 5). Setting the bit to 0 the "A" bank is used and setting the bit to 1 the "B" bank is used.

Attention: the use of the *_dscub_ParamSelector* variable is extremely important for the atomicity of the entire AO system. When the ARGOS system is driving the DM, the *_dscub_ParamSelector* variable should set by the ARGOS but the same variable is also passed to the DM for the use of the bit involved in the DM functioning. This approach assures a perfect atomicity of all the operations but requires a perfect synchronization of the activity of the ARGOS system in conjunction with the DM system. In particular for the bit #0, the same bit is also used for the parameters bank selection of the DM system, so if this bit is modified to swap the slope offset bank, also the second DM parameter banks should be updated accordingly.

The coefficient vector should be saved in *_dscfb_SlopeOffsetA* and *_dscfb_SlopeOffsetB* (pointed by *_dscub_SlopeOffsetAPtr* and *_dscub_SlopeOffsetAPtr* pointers) and organized as shown in Table 37 and Table 38.

DSP memory area	32 bit dword (float)
_dscfb_SlopeOffsetA	sub-ap #0 – slope x
_dscfb_SlopeOffsetA + 1	sub-ap #0 – slope y
_dscfb_SlopeOffsetA + 2	sub-ap #1 – slope x
_dscfb_SlopeOffsetA + 3	sub-ap #1 – slope y
_dscfb_SlopeOffsetA + 4	sub-ap #2 – slope x
_dscfb_SlopeOffsetA + 5	sub-ap #2 – slope y
……	……
_dscfb_SlopeOffsetA + 516	sub-ap #258 – slope x
_dscfb_SlopeOffsetA + 517	sub-ap #258 – slope y
_dscfb_SlopeOffsetA + 518	sub-ap #259 – slope x
_dscfb_SlopeOffsetA + 519	sub-ap #259 – slope y

Table 37 – Slope offset coefficients for DSP#0

DSP memory area	32 bit dword (float)
_dscfb_SlopeOffsetA	sub-ap #0 – slope x
_dscfb_SlopeOffsetA + 1	sub-ap #0 – slope y
_dscfb_SlopeOffsetA + 2	sub-ap #1 – slope x
_dscfb_SlopeOffsetA + 3	sub-ap #1 – slope y
_dscfb_SlopeOffsetA + 4	sub-ap #2 – slope x
_dscfb_SlopeOffsetA + 5	sub-ap #2 – slope y
……	……
_dscfb_SlopeOffsetA + 544	sub-ap #272 – slope x
_dscfb_SlopeOffsetA + 545	sub-ap #272 – slope y
_dscfb_SlopeOffsetA + 546	sub-ap #273 – slope x
_dscfb_SlopeOffsetA + 547	sub-ap #273 – slope y

Table 38 – Slope offset coefficients for DSP#1

8.2.6 Slope output

At this point the slope computation is terminated and the slope output vector is ready. The output is saved in *_dscfb_SlopeOutput* (pointed by *_dscub_SlopeOutputPtr* pointer) and organized as shown in Table 39 and Table 40.

DSP memory area	32 bit dword (float)
_dscfb_SlopeOutput	sub-ap #0 – slope x
_dscfb_SlopeOutput + 1	sub-ap #0 – slope y
_dscfb_SlopeOutput + 2	sub-ap #1 – slope x
_dscfb_SlopeOutput + 3	sub-ap #1 – slope y
_dscfb_SlopeOutput + 4	sub-ap #2 – slope x
_dscfb_SlopeOutput + 5	sub-ap #2 – slope y
……	……
_dscfb_SlopeOutput + 516	sub-ap #258 – slope x
_dscfb_SlopeOutput + 517	sub-ap #258 – slope y
_dscfb_SlopeOutput + 518	sub-ap #259 – slope x
_dscfb_SlopeOutput + 519	sub-ap #259 – slope y

Table 39 – Slope output vector for DSP#0

DSP memory area	32 bit dword (float)
_dscfb_SlopeOutput	sub-ap #0 – slope x
_dscfb_SlopeOutput + 1	sub-ap #0 – slope y
_dscfb_SlopeOutput + 2	sub-ap #1 – slope x
_dscfb_SlopeOutput + 3	sub-ap #1 – slope y
_dscfb_SlopeOutput + 4	sub-ap #2 – slope x
_dscfb_SlopeOutput + 5	sub-ap #2 – slope y
……	……
_dscfb_SlopeOutput + 544	sub-ap #272 – slope x
_dscfb_SlopeOutput + 545	sub-ap #272 – slope y
_dscfb_SlopeOutput + 546	sub-ap #273 – slope x
_dscfb_SlopeOutput + 547	sub-ap #273 – slope y

Table 40 – Slope output vector for DSP#1

Note: the previous vectors contain also the dummy slopes computed from the dummy centroid (the first 4 and the last 2 slopes). Here the dummy values are maintained and will be discarded by the DSP of the BCU board as described in 8.3.1.

Once that the entire computation is completed the *_dscub_SlopesCompleted* register is set to one to notify to the DSP of the BCU board that a new set of slopes is ready to be sent to the DM, see 8.3.

8.3 Mirrors update

Now that the slopes are calculated the new slope vector should be passed to the BCU-DSP device for the final activities, the mirrors update and the diagnostic storage.

8.3.1 Slope upload and reordering

The triggering of the end of the slope computation is done by the *_dscub_SlopesCompleted* register. After the end of the pixel download, the BCU-DSP device starts polling to that DSP register of DSP-ADC board until it is set to 1. With this notification, the BCU-DSP firmware launches the slope vectors transfer from both DSP devices of DSP-ADC boards.

According to the format described in Table 41, the data transfer puts the two vectors read from both DSP-ADC memory areas in the BCU-DSP memory area pointed by *_bscub_ReplyVectorPtr*. Note that the two vectors have the same size, in fact the command transfer is not able to read back from the two DSP devices two different buffer sizes.

DSP memory area	**32 bit dword (float)** **pnCCD fiber optic interface**	**32 bit dword (float)** **pnCCD direct on-board interface**
_bscub_ReplyVector	special word + 3 bytes	null
_bscub_ReplyVector + 1	frame number	null
_bscub_ReplyVector + 2	time stamp MSW	null
_bscub_ReplyVector + 3	time stamp LSW	null
_bscub_ReplyVector + 4	internal frame counter	internal frame counter
_bscub_ReplyVector + 5	_dscub_ParamSelector	_dscub_ParamSelector
_bscub_ReplyVector + 6	not used (0)	not used (0)
_bscub_ReplyVector + 7	not used (0)	not used (0)
_bscub_ReplyVector + 8	sub-ap #0 – slope x – DSP #0	sub-ap #0 – slope x – DSP #0
_bscub_ReplyVector + 9	sub-ap #0 – slope y – DSP #0	sub-ap #0 – slope y – DSP #0
_bscub_ReplyVector + 10	sub-ap #1 – slope x – DSP #0	sub-ap #1 – slope x – DSP #0
_bscub_ReplyVector + 11	sub-ap #1 – slope y – DSP #0	sub-ap #1 – slope y – DSP #0
_bscub_ReplyVector + 12	sub-ap #2 – slope x – DSP #0	sub-ap #2 – slope x – DSP #0
_bscub_ReplyVector + 13	sub-ap #2 – slope y – DSP #0	sub-ap #2 – slope y – DSP #0
……	……	……
_bscub_ReplyVector + 552	sub-ap #272 – slope x – DSP #0	sub-ap #272 – slope x – DSP #0
_bscub_ReplyVector + 553	sub-ap #272 – slope y – DSP #0	sub-ap #272 – slope y – DSP #0
_bscub_ReplyVector + 554	sub-ap #273 – slope x – DSP #0	sub-ap #273 – slope x – DSP #0
_bscub_ReplyVector + 555	sub-ap #273 – slope y – DSP #0	sub-ap #273 – slope y – DSP #0
_bscub_ReplyVector + 556	special word + 3 bytes	null
_bscub_ReplyVector + 557	frame number	null
_bscub_ReplyVector + 558	time stamp MSW	null
_bscub_ReplyVector + 559	time stamp LSW	null
_bscub_ReplyVector + 560	internal frame counter	internal frame counter
_bscub_ReplyVector + 561	_dscub_ParamSelector	_dscub_ParamSelector

_bscub_ReplyVector + 562	not used (0)	not used (0)
_bscub_ReplyVector + 563	not used (0)	not used (0)
_bscub_ReplyVector + 564	sub-ap #0 – slope x – DSP #1	sub-ap #0 – slope x – DSP #1
_bscub_ReplyVector + 565	sub-ap #0 – slope y – DSP #1	sub-ap #0 – slope y – DSP #1
_bscub_ReplyVector + 566	sub-ap #1 – slope x – DSP #1	sub-ap #1 – slope x – DSP #1
_bscub_ReplyVector + 567	sub-ap #1 – slope y – DSP #1	sub-ap #1 – slope y – DSP #1
……	……	……
_bscub_ReplyVector + 1107	sub-ap #272 – slope x – DSP #1	sub-ap #272 – slope x – DSP #1
_bscub_ReplyVector + 1108	sub-ap #272 – slope y – DSP #1	sub-ap #272 – slope y – DSP #1
_bscub_ReplyVector + 1109	sub-ap #273 – slope x – DSP #1	sub-ap #273 – slope x – DSP #1
_bscub_ReplyVector + 1111	sub-ap #273 – slope y – DSP #1	sub-ap #273 – slope y – DSP #1

Table 41 – Concatenated slope output vector from both DSPs

This read back vector contains all the slopes including the dummy slopes used to optimize the real-time computation and, when applicable, the not used slopes if any. From this vector the user has to extract and reorder the used slopes and put them in the final vector *_bscfb_SlopeVector* (pointed by *_bscub_SlopeVectorPtr* pointer) that is sent to the secondary mirror. This operation is very important because the used slopes and in particular the slope order should match the reconstructor parameters setup.

The final number of slopes is defined for each LGS setting the variables *_bscub_NumSlopesLGS0*, *_bscub_NumSlopesLGS1* and *_bscub_NumSlopesLGS2* according to the excel spread sheet (see [RD1]).

Instead, to extract the slopes from the *_bscub_ReplyVector* and put correctly in the *_bscfb_SlopeVector* area, the *_bscub_RemapSlopeVector* (which size is *_bscub_NumSlopesLGS0 + _bscub_NumSlopesLGS1 + _bscub_NumSlopesLGS2*) should be initialized as described in Table 42.

DSP memory area	32 bit dword (uint32)
_bscub_RemapSlopeVector	slope #0 – LGS #0
_bscub_RemapSlopeVector + 1	slope #1 – LGS #0
_bscub_RemapSlopeVector + 2	slope #2 – LGS #0
_bscub_RemapSlopeVector + 3	slope #3 – LGS #0
……	……
_bscub_RemapSlopeVector + n_{LGS0} – 2	slope #(n_{LGS0} – 2) – LGS #0
_bscub_RemapSlopeVector + n_{LGS0} – 1	slope #(n_{LGS0} – 1) – LGS #0
_bscub_RemapSlopeVector + n_{LGS0}	slope #0 – LGS #1
_bscub_RemapSlopeVector + n_{LGS0} + 1	slope #1 – LGS #1
……	……
_bscub_RemapSlopeVector + n_{LGS0} + n_{LGS1} – 2	slope #(n_{LGS1} – 2) – LGS #1
_bscub_RemapSlopeVector + n_{LGS0} + n_{LGS1} – 1	slope #(n_{LGS1} – 1) – LGS #1
_bscub_RemapSlopeVector + n_{LGS0} + n_{LGS1}	slope #0 – LGS #2
_bscub_RemapSlopeVector + n_{LGS0} + n_{LGS1} + 1	slope #1 – LGS #2
……	……
_bscub_RemapSlopeVector + n_{LGS0} + n_{LGS1} + n_{LGS2} – 2	slope #(n_{LGS1} – 2) – LGS #2
_bscub_RemapSlopeVector + n_{LGS0} + n_{LGS1} + n_{LGS2} – 1	slope #(n_{LGS1} – 1) – LGS #2

Table 42 – Final slope vector remapping vector

8.3.2 LGS tip-tilt slopes computation and TT mirrors update

During the slope reordering, the DSP also computes the LGS tip-tilt slopes as the mean (note: in [AD1] was required the median but we modified the algorithm in accordance with Gilles Orban de Xivry) of all slopes for each LGS. Completed the slopes reorder and consequently the LGS tip-tilt computation, the six slopes are sent to the two HVC boards for the field pointing tip-tilt mirrors update (see 8.4.6 for the description of the HVC firmware functioning).

The data transfer uses the high speed link and should be properly initialized; see 8.4.8 for the description of all high speed link commands used in ARGOS design.

8.3.3 Secondary mirror update

Immediately after the TT mirrors update the slope vector is sent to the secondary mirror to execute the reconstructor algorithm and update the DM commands.

The slope vector written to the DM should be compliant with the reconstructor format. As already mentioned the vector is created in *_bscfb_SlopeVector* and should have a fixed size of 1600 elements (for slopes) and a footer of 4 dwords as described in Table 43. In the same vector there is also the collecting of the MPIfR TT slopes (see 8.4.4) and eventually the FLAO and/or BCU-Na slopes (see 8.4.5). The not used slopes to fill the 1600 elements are set to zero. The footer (pointed by *_bscub_StartRTRPtr*) contains the *_dscub_ParamSelector* word for the atomic system update of the parameters and activities and the *_dscub_FramesCounter* to identify the reconstructor outputs and DM commands with the input slopes. The last word of the footer is the triggering register to start the reconstructor algorithm.

DSP memory area	32 bit dword (float + uint32)
_bscfb_SlopeVector	pnCCD slope #0
_bscfb_SlopeVector + 1	pnCCD slope #1
……	……
_bscfb_SlopeVector + n_{pnCCD} – 2	pnCCD slope #(n_{pnCCD} – 2)
_bscfb_SlopeVector + n_{pnCCD} – 1	pnCCD slope #(n_{pnCCD} – 1)
_bscfb_RotTTSlopeVect	MPIfR TT slope #0
_bscfb_RotTTSlopeVect + 1	MPIfR TT slope #1
……	……
_bscfb_RotTTSlopeVect + n_{TT} – 2	MPIfR TT slope #(n_{TT} – 2)
_bscfb_RotTTSlopeVect + n_{TT} – 1	MPIfR TT slope #(n_{TT} – 2)
_bscfb_FlaoNaSlopeVect	FLAO_Na slope#0
_bscfb_FlaoNaSlopeVect + 1	FLAO_Na slope#1
……	……
_bscfb_FlaoNaSlopeVect + n_{FLAO_Na} – 2	FLAO_Na slope#(n_{FLAO_Na} – 2)

_bscfb_FlaoNaSlopeVect + n_{FLAO_Na} – 1	FLAO_Na slope#(n_{FLAO_Na} – 1)
_bscfb_FlaoNaSlopeVect + n_{FLAO_Na} ……	not used slopes
_bscub_StartRTR	_dscub_FramesCounter
_bscub_StartRTR + 1	_dscub_ParamSelector
_bscub_StartRTR + 2	not used (0)
_bscub_StartRTR + 3	1

Table 43 – Final slope vector to send to DM

The data transfer of this vector uses the high speed link and should be properly initialized; see 8.4.8 for the description of all high speed link commands used in ARGOS design.

8.4 Real time diagnostic storage

Completed the real time computational part, both DSP-ADC and BCU boards start with the diagnostic record creation and storing. The DSP-ADC board stores the pnCCD pixel frame while the BCU board stores the slope data record.

8.4.1 pnCCD pixel frame diagnostic storage

As mentioned, completed the real time computational part, the DSP device launches an automatic mechanism for the storage of the pnCCD pixel frame. The entire frame is read from the *_dscub_PixelArea* (pointed by *_dscub_PixelAreaPtr* pointer) of the DSP device memory and copied in the SDRAM bulk memory contiguously to the previous pixel frame. The data format of the pixel frame has the same format as reported in Table 17 apart for the first two locations where the internal frame counter (*_dscub_FramesCounter*) and the *_dscub_ParamSelector* register are saved, see the right column of the table.

Once the end of the SDRAM is reached and there is not enough space for another entire record the system restarts saving the record from the zero position.

The bits to transfer are (264*264+8)*16=1.063Mbit and using the diagnostic link at 2Gbit/s it requires about 0.5ms. Considering the available SDRAM on the DSP-ADC board, the number of frames that can be saved on the SDRAM is 960 before to generate the wrap-around of the buffer.

The saved frames can be decimated using the *_dscub_DiagnosticFrameDec* according to the mechanism described in 8.4.3.1.

As requested in [AD1] there are two different frames upload,

- the first one is the off-loading upload used to have the frames at full (or decimated) rate, on this case the user has to stop the diagnostic storage before and read back all (or one part) of the DSP-ADC SDRAM memory and form the read buffer he has extract all the frames saved.

- the second method is used to obtain the pixel frame at low frequency but in real-time mode during the system activity. The maximum frequency upload depends on the bandwidth of the service link balanced to the system load at currently is set at 50 Hz. To set a decimated pixel frame storing the user has to set the *_dscub_DiagnosticFrameDec* variable as described in 8.4.3.1. This method works in conjunction with the slope data record storage as described in 8.4.2.1.

8.4.2 Diagnostic slope data record storage

Also for the BCU board, once the real time update of the mirrors is completed, the DSP device creates the diagnostic record in the DSP memory area and launches an automatic mechanism for the storage of the record to the SDRAM bulk memory. The record starts at the DSP memory pointed by *_bscub_HeaderDiagPtr* pointer and ends at the DSP memory pointed by *_bscub_FooterDiagPtr* + 4 (4 is the size of the *_bscub_FooterDiag* vector), the size of the record is fixed as fixed are all the subparts of the record and it is 1628 dwords. Table 44 shows the structure of the diagnostic record.

The record is stored to the BCU–SDRAM memory in sequence respect to the previous one. Once the end of the SDRAM is reached and there is not enough space for another entire record the system restarts saving the record from the zero position.

Considering the fastest frame rate of 1kHz, the total transfer rate required by the stored data is 1656*32*1000 = 50.537Mbit/s; these data are transferred to the ARGOS supervisor via 1Gbit/s Ethernet link. Even if the bandwidth is absolutely not critical, a dedicated data transfer mechanism (called master-BCU, see 8.4.3.2) is available in order to optimize the data transfer and reduce the transfer load.

As requested in [AD1] also the pnCCD pixel frames should be downloaded to the ARGOS supervisor in real time at low frequency. We can assume a reasonable Ethernet transfer load of ~100Mbit/s (in master-BCU mode), so the maximum acceptable pnCCD pixel frame rate at real time download is 50Hz, in fact (1656*32)*1000 + (264*264*16+8)*50 = 103.711Mbit/s. This rate is double w.r.t. the requested rate in [AD1].

The Table 44 shows the structure of the real-time diagnostic record with a link to the tables of the various sub-blocks.

DSP memory area	32 bit dword (float + uint32)	Size (dwords)
_bscub_HeaderDiag	_dscub_FramesCounter	4
_bscub_HeaderDiag + 1	_dscub_ParamSelector	
_bscub_HeaderDiag + 2	not used (0)	
_bscub_HeaderDiag + 3	not used (0)	
_bscfb_SlopeVector	n_{pnCCD} pnCCD slopes	1600
_bscfb_RotTTSlopeVect	n_{TT} MPIfR TT slopes	
_bscfb_FlaoNaSlopeVect	n_{FLAO_Na} FLAO_Na slopes	
……	not used area	
_bscub_StartRTR	see Table 43	4
_bscfb_TTSlopesVector	see Table 45	8

_bscfb_HVCMeanVoltage	see Table 46	16
_bscfb_HVCMeanPosition	see Table 47	16
_bscub_APDCounters	see Table 48	4
_bscub_FCVector	see Table 49	8
_bscub_FooterDiag	same of _bscub_HeaderDiag	4

Table 44 – Real-time diagnostic record

The *_bscfb_TTSlopesVector* is a fixed size vector of 8 elements containing the compute LGS tip-tilt slopes, the format is described in Table 45.

DSP memory area	32 bit dword (float)
_bscfb_TTSlopesVector	TT slope x of LGS #0
_bscfb_TTSlopesVector + 1	TT slope y of LGS #0
_bscfb_TTSlopesVector + 2	TT slope x of LGS #1
_bscfb_TTSlopesVector + 3	TT slope y of LGS #1
_bscfb_TTSlopesVector + 4	TT slope x of LGS #2
_bscfb_TTSlopesVector + 5	TT slope y of LGS #2
_bscfb_TTSlopesVector + 6	not used (0)
_bscfb_TTSlopesVector + 7	not used (0)

Table 45 – LGS tip-tilt slopes diagnostic record

The *_bscfb_HVCMeanVoltage* is a fixed size vector of 12 elements containing the mirror actuator averaged (during the integration time) set voltage of the piezoelectric tip-tilt mirrors, the format is described in Table 46.

DSP memory area	32 bit dword (float)
_bscfb_HVCMeanVoltage	channel #0 voltage (tip-tilt #0 of HVC board #0)
_bscfb_HVCMeanVoltage + 1	channel #1 voltage (tip-tilt #0 of HVC board #0)
_bscfb_HVCMeanVoltage + 2	channel #2 voltage (tip-tilt #0 of HVC board #0)
_bscfb_HVCMeanVoltage + 3	not used (0)
_bscfb_HVCMeanVoltage + 4	channel #0 voltage (tip-tilt #1 of HVC board #0)
_bscfb_HVCMeanVoltage + 5	channel #1 voltage (tip-tilt #1 of HVC board #0)
_bscfb_HVCMeanVoltage + 6	channel #2 voltage (tip-tilt #1 of HVC board #0)
_bscfb_HVCMeanVoltage + 7	not used (0)
_bscfb_HVCMeanVoltage + 8	channel #0 voltage (tip-tilt #0 of HVC board #1)
_bscfb_HVCMeanVoltage + 9	channel #1 voltage (tip-tilt #0 of HVC board #1)
_bscfb_HVCMeanVoltage + 10	channel #2 voltage (tip-tilt #0 of HVC board #1)
_bscfb_HVCMeanVoltage + 11	not used (0)
_bscfb_HVCMeanVoltage + 12	channel #0 voltage (tip-tilt #1 of HVC board #1)
_bscfb_HVCMeanVoltage + 13	channel #1 voltage (tip-tilt #1 of HVC board #1)
_bscfb_HVCMeanVoltage + 14	channel #2 voltage (tip-tilt #1 of HVC board #1)
_bscfb_HVCMeanVoltage + 15	not used (0)

Table 46 –Tip-tilt mirrors output voltages diagnostic record

The *_bscfb_HVCMeanPosition* is a fixed size vector of 12 elements containing the mirror actuator average position (during the integration time) read by the strain gauges of the piezoelectric tip-tilt mirrors, the format is described in Table 47.

DSP memory area	32 bit dword (float)
_bscfb_HVCMeanPosition	channel #0 position (tip-tilt #0 of HVC board #0)
_bscfb_HVCMeanPosition + 1	channel #1 position (tip-tilt #0 of HVC board #0)
_bscfb_HVCMeanPosition + 2	channel #2 position (tip-tilt #0 of HVC board #0)
_bscfb_HVCMeanPosition + 3	not used (0)
_bscfb_HVCMeanPosition + 4	channel #0 position (tip-tilt #1 of HVC board #0)
_bscfb_HVCMeanPosition + 5	channel #1 position (tip-tilt #1 of HVC board #0)
_bscfb_HVCMeanPosition + 6	channel #2 position (tip-tilt #1 of HVC board #0)
_bscfb_HVCMeanPosition + 7	not used (0)
_bscfb_HVCMeanPosition + 8	channel #0 position (tip-tilt #0 of HVC board #1)
_bscfb_HVCMeanPosition + 9	channel #1 position (tip-tilt #0 of HVC board #1)
_bscfb_HVCMeanPosition + 10	channel #2 position (tip-tilt #0 of HVC board #1)
_bscfb_HVCMeanPosition + 11	not used (0)
_bscfb_HVCMeanPosition + 12	channel #0 position (tip-tilt #1 of HVC board #1)
_bscfb_HVCMeanPosition + 13	channel #1 position (tip-tilt #1 of HVC board #1)
_bscfb_HVCMeanPosition + 14	channel #2 position (tip-tilt #1 of HVC board #1)
_bscfb_HVCMeanPosition + 15	not used (0)

Table 47 – Tip-tilt mirrors mean position diagnostic record

The *_bscub_APDcounters* is a fixed size vector of 4 elements containing the APD counters received from the last MPIfR serial data packet, the format is described in Table 48.

DSP memory area	32 bit dword (uint16)
_bscub_APDCounters	APD counter #1
_bscub_APDCounters + 1	APD counter #2
_bscub_APDCounters + 2	APD counter #3
_bscub_APDCounters + 3	APD counter #4

Table 48 – APD counters diagnostic record

The *_bscub_FCVector* is a fixed size vector of 4 elements with the collecting of all frame counters and the time stamp of the last pnCCD pixel frame, the format is described in Table 49.

DSP memory area	32 bit dword (uint32) pnCCD fiber optic interface	32 bit dword (uint32) pnCCD direct on-board interface
_bscub_FCVector	_dscub_FrameCounter	_dscub_FrameCounter
_bscub_FCVector + 1	pnCCD frame number	pnCCD frame number

_bscub_FCVector + 2	time stamp MSW	null
_bscub_FCVector + 3	time stamp LSW	null
_bscub_FCVector + 4	MPIfR frame number	MPIfR frame number
_bscub_FCVector + 5	MPIfR status byte	MPIfR status byte
_bscub_FCVector + 6	flao_FramesCounter	flao_FramesCounter
_bscub_FCVector + 7	na_FramesCounter	na_FramesCounter

Table 49 – Frames number diagnostic record

8.4.2.1 Combined slope and pixels data record storage

As requested in [AD1] the system should be able to store and collect the slope data record in parallel with a decimated set of pixel frames. To do this we introduced a second flag (bit #12) in the *_dscub_ParamSelector* word (see Table 5) to enable this second level of storage mechanism, in fact the first flag (bit #6) enables just the "local" diagnostic storage of the slope records in the BCU board and the pixel frames in the DSP-ADC board.

If the bit #12 of *_dscub_ParamSelector* word is enabled ALL the pixels frames saved in the local DSP-ADC SDRAM memory (according to the *_dscub_DiagnosticFrameDec*) are also transferred to the BCU SDRAM memory. For this reason, when the bit #12 of *_dscub_ParamSelector* word is enabled, the setting of the decimation counter word should be set carefully according to the maximum data bandwidth.

The format of the pixels frame depends to the pnCCD interface currently used. In particular Table 17 describes the frame format if the fiber optic interface is used and Table 18 describes the frame format when the direct analog acquisition is adopted.

Figure 14 shows how the records are stored in the BCU SDRAM, with a diagnostic configuration of no decimation on data slope records (*_bscub_DiagnosticFrameDec* = 0) and a decimated pixel frame of 20 (1 pixel frame every 20 frames, *_dscub_DiagnosticFrameDec* = 19).

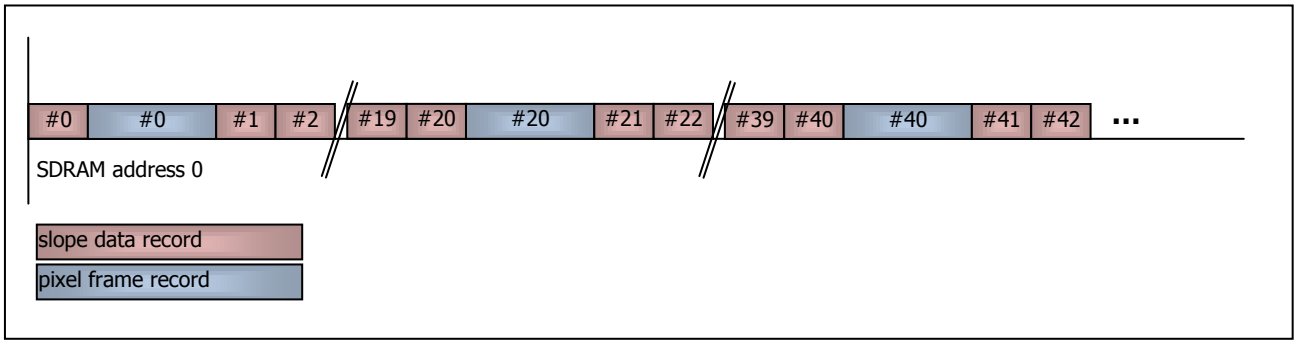

Figure 14 – diagnostic data records on BCU SDRAM memory

8.4.3 Real time diagnostic storage register initialization

Before enabling the real time diagnostic storage and eventually the master-BCU mechanism some register of both BCU Nios firmware and DSP-ADC Nios firmware must be initialized to save the data properly. The registers to initialize in the SDRAM are the following:

Variable name	Address (dword)	Type	Description
diagnostic_record_ptr	0x000181F5	uint32	Pointer to the DSP memory where to read the diagnostic record (refer to *_dscub_PixelAreaPtr* in 8.4.1)
diagnostic_record_len	0x000181F8	uint32	Diagnostic record size in DWORD (refer to 8.4.1)
rd_diagnostic_record_ptr	0x000181F7	uint32	Read only: current pointer for internal use
wr_diagnostic_record_ptr	0x000181F8	uint32	Read only: current pointer to the SDRMA where the diagnostic data read from the DSP is stored

Table 50 – Diagnostic storage DSP-ADC control registers

Variable name	Address (dword)	Type	Description
diagnostic_record_ptr	0x00038012	uint32	Pointer to the DSP memory where to read the diagnostic record (refer to *_bscub_HeaderDiagPtr*)
diagnostic_record_len	0x00038013	uint32	Diagnostic record size in DWORD (refer to 8.4.2)
enable_master_diag	0x00038014 (LSW)	uint16	Enable (1) or disable (0) the master-BCU mechanism
decimation_factor	0x00038014 (MSW)	uint16	Decimation factor of the frame sent by the BCU via master-BCU mechanism
remote_mac_address	0x00038015 0x00038016 (LSW)	6 x uint8	MAC address of the remote server where to send the packets (ARGOS supervisor)
remote_ip_address	0x00038016 (MSW) 0x00038017 (LSW)	4 x uint8	IP address of the remote server where to send the packets (ARGOS supervisor)
remote_udp_port	0x00038017 (MSW)	uint16	UDP port of the remote server where to send the packets (ARGOS supervisor)
rd_diagnostic_record_ptr	0x00038018	uint32	Read only: current pointer to the SDRMA where the master-BCU downloading has arrived
wr_diagnostic_record_ptr	0x00038019	uint32	Read only: current pointer to the SDRMA where the diagnostic data read from the DSP is stored

Table 51 – Diagnostic storage BCU control registers

Once the registers have been initialized, including the *_dscub_DiagnosticFrameDec* and *_bscub_DiagnosticFrameDec* variables, it is possible to enable the diagnostic storage setting to one the proper bit of *_dsubc_ParamSelector* register (see Table 5).

8.4.3.1 RT diagnostic decimation factor

Setting properly the *_dscub_DiagnosticFrameDec* and *_bscub_DiagnosticFrameDec* DSP variable, one can enable a decimation mechanism to reduce the number of stored frames, based on the module of the frame

counter. By default the decimation factor is 0 that means save all frames and Table 53 explains how the decimation mechanism works.

Frame number	Decimation Factor							
	0	1	2	3	4	5	6	7
0	yes	yes	yes	yes	yes	yes	yes	yes
1	yes							
2	yes	yes						
3	yes		yes					
4	yes	yes		yes				
5	yes				yes			
6	yes	yes	yes			yes		
7	yes						yes	
8	yes	yes		yes				yes
9	yes		yes					
10	yes	yes			yes			
11	yes							
12	yes	yes	yes	yes		yes		
13	yes							
14	yes	yes					yes	

Table 52 – Decimeation factor vs Frame number

Decimation Factor	0	all frames
	1	1 frame yes and 1 no
	2	1 frame yes and 2 no
	3	1 frame yes and 3 no
	4	1 frame yes and 4 no
	5	1 frame yes and 5 no
	6	1 frame yes and 6 no
	7	1 frame yes and 7 no

Table 53 – Decimation factor list

8.4.3.2 Master-BCU mechanism

The master-BCU mechanism is an automatic mechanism that sends to an external machine (in our case the ARGOS supervisor) the diagnostic records directly from the BCU to the supervisor, without implementing a bidirectional communication protocol. In fact the standard communication Ethernet link is based on a strictly client/server architecture, where the client (ARGOS supervisor) has to send a request of data to the BCU (server) and it responses with the requested data (see also 8.4.9). This approach is valid for all the standard communication activities where the real time performances are not requested, while in case the system has to upload huge amounts of data this approach is not the most efficient and for this reason the master-BCU mechanism has been implemented. With this mechanism the supervisor has to initialize the IP

and MAC addresses where the BCU has to send the data, enable the mechanism (see Table 51 for the register initialization) and as soon as a new record is available in the BCU-SDRAM memory the BCU sends the frame to the remote machine. Excluding the standard ETH/IP/UDP header and footer, the maximum Ethernet packet size is 1470 (14 bytes of header + 1456 of payload) (see Table 68) then the entire record is divided in a certain number of sub frames. The Ethernet protocol adopted is a UDP-IP with an encapsulated custom packet called TDP. The packet has a header with the following structure:

```
// header of master-BCU data packets
struct tdphdr
      {
      uint16  dummy_word;  // dummy word for 32 bit alignment
      uint32  tot_len;     // total length of the diagnostic record (in bytes)
      uint32  saddr;       // start address of this sub-packet respect to the entire record
                              (starting from 0)
      uint32  id;          // LSW record id, MSW sub-record id
      };
```

The supervisor has to open an UDP socket on the predefined port and to start a listening process that waits and captures all the arriving sub-packets and from them has to rebuild the data record.

This method is extremely efficient reducing the CPU and Ethernet link load and should be adopted for the diagnostic acquisition.

8.4.4 MPIfR frame acquisition and managing

The ARGOS BCU system is able to receive and manage the serial frames sent by the MPIfR system according to the protocol defined in [AD7]. The packet is received from the serial interface and forwarded to the shared memory area of the BCU-DSP memory *_bscfb_ExtRotTTSlopeVect*. Once the entire packet has arrived (an additional <cr> character has been added to detect the end of packet) the serial interface notifies to the DSP that the new packet has arrived. First the DSP firmware checks the validity of the checksum and if it is valid, arranges the input data in the following way:

- puts frame number in *_bscub_FCVector* + 4
- puts x & y coordinates in *_bscfb_RotTTSlopeVect* converting it in floating point single precision format (note that in the slope vector the initialization procedure reserves 4 dwords even if just two locations are needed, the other two dwords are not used (set to 0) but should be left for internal code optimization). Another point is that no operation is done on these slopes (neither gain nor offset correction) and they are just passed, after conversion to floating point, to the reconstructor as they are.
- puts the four APD counters in *_bscub_APDCounters*, leaving them in uint16 bit format but extended in four 32bit dwords.
- once the last arrived set of TT slopes has been sent to the DM, the slopes are reset to zero waiting for a new set of slopes from the detector, it means that each set of TT slopes is sent to the DM

once and between one set and the following one all the other slope vector sent to the DM are filled to 0 for what concern the *_bscfb_RotTTSlopeVect* area.

The firmware has also a mechanism to avoid partial overwriting during the slope or diagnostic data transfer. During those phases, the arriving data is temporary left in the shared memory (*_bscfb_ExtRotTTSlopeVect*) and only when the two critical phases are completed, the frame is processed.

8.4.5 FLAO and Na slopes acquisition and managing

Similar to the MPIfR interface, the ARGOS BCU system is able to receive and manage slope vectors arriving from either FLAO BCU or Na BCU. In that case the interface used is the high speed link which writes the slope vector in the proper shared memory. The configuration of the command should be done in the FLAO BCU or NaBCU, which must write the vector into the *_bscfb_ExtFlaoNaSlopeVect* area (pointed by the *_bscub_ExtFlaoNaSlopeVectPtr* pointer). The frame must have the standard format used to transfer data between the slope computers and the DM, as described in Table 54 where the only FLAO system is present (option 1 of Figure 1 of [AD1]) and Table 55 where both FLAO and Na BCU systems are present (option 2 of Figure 1 of [AD1]).

DSP memory area	**32 bit dword (float + uint32)**
_bscfb_ExtFlaoNaSlopeVect	FLAO_Na slope#0
_bscfb_ExtFlaoNaSlopeVect + 1	FLAO_Na slope#1
……	……
_bscfb_ExtFlaoNaSlopeVect + $n_{FLAO\_Na} - 2$	FLAO_Na slope#($n_{FLAO\_Na} - 2$)
_bscfb_ExtFlaoNaSlopeVect + $n_{FLAO\_Na} - 1$	FLAO_Na slope#($n_{FLAO\_Na} - 1$)
_bscub_ExtFlaoNaStartRTR	flao_FramesCounter
_bscub_ExtFlaoNaStartRTR + 1	flao_ParamSelector
_bscub_ExtFlaoNaStartRTR + 2	not used (0)
_bscub_ExtFlaoNaStartRTR + 3	1

Table 54 – FLAO solo slope vector data format

DSP memory area	**32 bit dword (float + uint32)**
_bscfb_ExtFlaoNaSlopeVect	FLAO_Na slope#0
_bscfb_ExtFlaoNaSlopeVect + 1	FLAO_Na slope#1
……	……
_bscfb_ExtFlaoNaSlopeVect + $n_{FLAO\_Na} - 2$	FLAO_Na slope#($n_{FLAO\_Na} - 2$)
_bscfb_ExtFlaoNaSlopeVect + $n_{FLAO\_Na} - 1$	FLAO_Na slope#($n_{FLAO\_Na} - 1$)
_bscub_ExtFlaoNaStartRTR	flao_FramesCounter
_bscub_ExtFlaoNaStartRTR + 1	na_ParamSelector
_bscub_ExtFlaoNaStartRTR + 2	na_FramesCounter
_bscub_ExtFlaoNaStartRTR + 3	1

Table 55 – FLAO + Na slope vector data format

When a new FLAO + Na slope frame has arrived, triggered by the *_bscub_ExtFlaoNaStartRTR* + 3 register set to 1, the DSP firmware arranges the input data in the following way:

- puts *flao_FramesCounter* in *_bscub_FCVector* + 5 and *na_FramesCounter* in *_bscub_FCVector* + 6;
- copies the slope vector in *_bscfb_FlaoNaSlopeVect*.

Also in this case the firmware has a mechanism to avoid partial overwriting during the slope or diagnostic data transfer. During those phases, the arriving data is temporary left in the shared memory (*_bscfb_ExtFlaoNaSlopeVect*) and only when the two critical phases are completed, the frame is processed.

Attention: if either FLAO BCU or Na BCU is used, the fast link command generated by FLAO/Na that writes the slopes into *_bscfb_ExtFlaoNaSlopeVect* memory area must be modified with respect to the standard value, in fact the ARGOS interface requires the generation of the interrupt to the receiver DSP (ArgosBCU DSP). To do this the bit #14 of the first command register of the FLAO/Na Slope Computer command should be set to 1 (see Table 62).

8.4.6 HVC control board

The aim of the HCV board is to control the three LGS tip-tilt piezoelectric mirrors. The commands are generated from the pnCCD slopes as described in 8.3.2. Each board is able to drive up to two tip-tilt mirrors so the ARGOS BCU mini-crate is provided by two HVC board one fully used and the other half used. The tip-tilt command vector computed by the BCU-DSP device has 8 elements as described in Table 56 and it is transferred to both HVC boards in the HVC-DSP shared memory *_hvcfb_TTCommandVector* using a FastLink commands (see 8.4.8).

The original computational design has been modified according to the request by Lorenzo Busoni to introduce a pure integrator in the algorithm. The following description has been updated according to this request.

DSP memory area	32 bit dword (float + uint32)
_hvcfb_TTCommandVector	x command – LGS #0
_hvcfb_TTCommandVector + 1	y command – LGS #0
_hvcfb_TTCommandVector + 2	x command – LGS #1
_hvcfb_TTCommandVector + 3	y command – LGS #1
_hvcfb_TTCommandVector + 4	x command – LGS #2
_hvcfb_TTCommandVector + 5	y command – LGS #2
_hvcfb_TTCommandVector + 6	not used (0)
_hvcfb_TTCommandVector + 7	trigger dword (1)

Table 56 – LGS tip-tilt command vector

As soon as a new vector has arrived to the HVC-DSP device (triggered by the last element of the vector), the main routine of the HVC firmware selects for each tip-tilt mirror the corresponding commands. To do this a selection matrix with 2x8 elements should be properly initialized (variables *_hvcfb_SelectionMatrixTT0* and *_hvcfb_SelectionMatrixTT1*) as shown in Table 57 (same for TT1 mirror) which will be multiplied by the *_hvcfb_TTCommandVector* to extract the tip and tilt command for each mirror.

DSP memory area	32 bit dword (float)
_hvcfb_SelectionMatrixTT0	tip command extraction
_hvcfb_SelectionMatrixTT0 + 1	
_hvcfb_SelectionMatrixTT0 + 2	
_hvcfb_SelectionMatrixTT0 + 3	
_hvcfb_SelectionMatrixTT0 + 4	
_hvcfb_SelectionMatrixTT0 + 5	
_hvcfb_SelectionMatrixTT0 + 6	
_hvcfb_SelectionMatrixTT0 + 7	
_hvcfb_SelectionMatrixTT0 + 8	tilt command extraction
_hvcfb_SelectionMatrixTT0 + 9	
_hvcfb_SelectionMatrixTT0 + 10	
_hvcfb_SelectionMatrixTT0 + 11	
_hvcfb_SelectionMatrixTT0 + 12	
_hvcfb_SelectionMatrixTT0 + 13	
_hvcfb_SelectionMatrixTT0 + 14	
_hvcfb_SelectionMatrixTT0 + 15	

Table 57 – TT0 selection matrix data format

After that, using a 2x2 rotation and gain matrix, the two x and y angular mirror commands can be rotated and scaled (in radiant). The matrix is saved in the *_hvcfb_RotationMatrixTT0* and *_hvcfb_RotationMatrixTT1* variables (see Table 58 for the matrix data format, same for TT1 mirror).

DSP memory area	32 bit dword (float)
_hvcfb_RotationMatrixTT0	mirror x angle – coefficient of element tip
_hvcfb_RotationMatrixTT0+ 1	mirror x angle – coefficient of element tilt
_hvcfb_RotationMatrixTT0+ 2	mirror y angle – coefficient of element tip
_hvcfb_RotationMatrixTT0+ 3	mirror y angle – coefficient of element tilt

Table 58 – TT0 rotation matrix data format

Now, since these two angles are a delta command, they should be added to the previous command to obtain the new final command. The integrator is a pure integrator with unitary gain, the integrated value is stored to the variable *_hvcfb_IntegratedCommandTT0* and *_hvcfb_IntegratedCommandTT1.*

If a reset of the integrated value is needed the firmware includes a dedicated variable called *_hvcub_ResetIntegratorTT0*. If this variable is set by the host to true (>0) the integrator is reset setting the variable *_hvcfb_IntegratedCommandTT0* to zero, at the same time the firmware automatically reset to zero also the variable *_hvcub_ResetIntegratorTT0*. Same mechanism is also available for the TT1 mirror.

Then the integrated command can be corrected by an offset for each axis (variables *_hvcfb_CommandOffsetTT0*) as described in Table 59 (same for TT1 mirror). For example the offset can be used to set the zero position of the mirror according to the optical alignment.

DSP memory area	32 bit dword (float)
_hvcfb_CommandOffsetTT0	tip command offset
_hvcfb_CommandOffsetTT0 + 1	tilt command offset

Table 59 – TT0 command offset vector

The routine performs two different command checks: first the angular mirror commands are checked with respect to a minimum and maximum range (set in *_hvcfb_MinCommandTT0*, *_hvcfb_MaxCommandTT0*, *_hvcfb_MinCommandTT1* and *_hvcfb_MaxCommandTT1*) and eventually clipped to those values. The second check is done on the delta command of the new command with respect to the previous one, if the delta command is higher than *_hvcfb_MaxDeltaCommand*, the new command is clipped to this maximum allowable delta command.

The last computational step is the feed forward voltage computation using the *_hvcfb_FFGainTT0* and *_hvcfb_FFGainTT1* (see Table 60 for the matrix data format, same for TT1 mirror). As described in 8.4.6.1, the aim of this contribution is fundamental to increase the dynamic performances of the local control loop and it is computed as a proportional gain of the actuator commanded position.

DSP memory area	32 bit dword (float)
_hvcfb_FFGainTT0	FF gain for actuator channel #0
_hvcfb_FFGainTT0 + 1	FF gain for actuator channel #1

Table 60 – TT0 feedforward gain vector

Before passing the new angular commands to the actuator control loop a consideration should be done regarding the tip-tilt mirror (PI-S-334.1SL) adopted for the ARGOS. In fact this device is controlled driving actively only two channels, while a third channel is driven at constant voltage (typically set at mid output range). In the ARGOS system the setting of the third static channel voltage is obtained using the

_hvcfb_TT0_bias_current and *_hvcfb_TT1_bias_current* variables with the proper voltage to DAC bit conversion parameter that will be calibrated on each specific tip tilt mirror.

The new angular commands are passed to the local control loop copying the angular commands in the variable labeled as *pos_command* in Figure 15 (the name in the HVC firmware is *_hvcfb_command_vector*, see [RD1] - sheet *HVCMainProgram*) and the two feed forward voltages to the variable labeled as *ff_ud_current* in Figure 15 (the name in the HVC firmware is *_hvcfb_current_vector*, see [RD1] - sheet *HVCMainProgram*), and then processed by the local control loop using a dedicated trigger word (see 8.4.6.1).

8.4.6.1 HVC Local actuator servo control loop

The local actuator servo control loop is an interrupt service routine @ ~70 KHz and its scope is to control the actuators of the tip-tilt mirrors in position.

The routine implements a digital IIR filter with 3 taps both on the position error and on the absolute position. This is typically used to implement a derivative control acting as ‘electronic damper’. The filter is completely independent for each mirror actuator.

The output is converted to voltage and sent to the HV driver and then to the corresponding tip-tilt mirror actuator.

The feedback actuator position is acquired by the strain gauge available on the tip-tilt mirror.

In parallel with the position close loop a command proportional open loop contribution is included in the algorithm, this contribution is important to obtain the maximum performances at the step response where the close loop is relatively slow while this output proportional contribution allows at the actuator to reach the final position as fast as possible.

Input and output gains and offsets on each actuator allow considering strain gauge and actuator calibration, so that the control filter has unity gain.

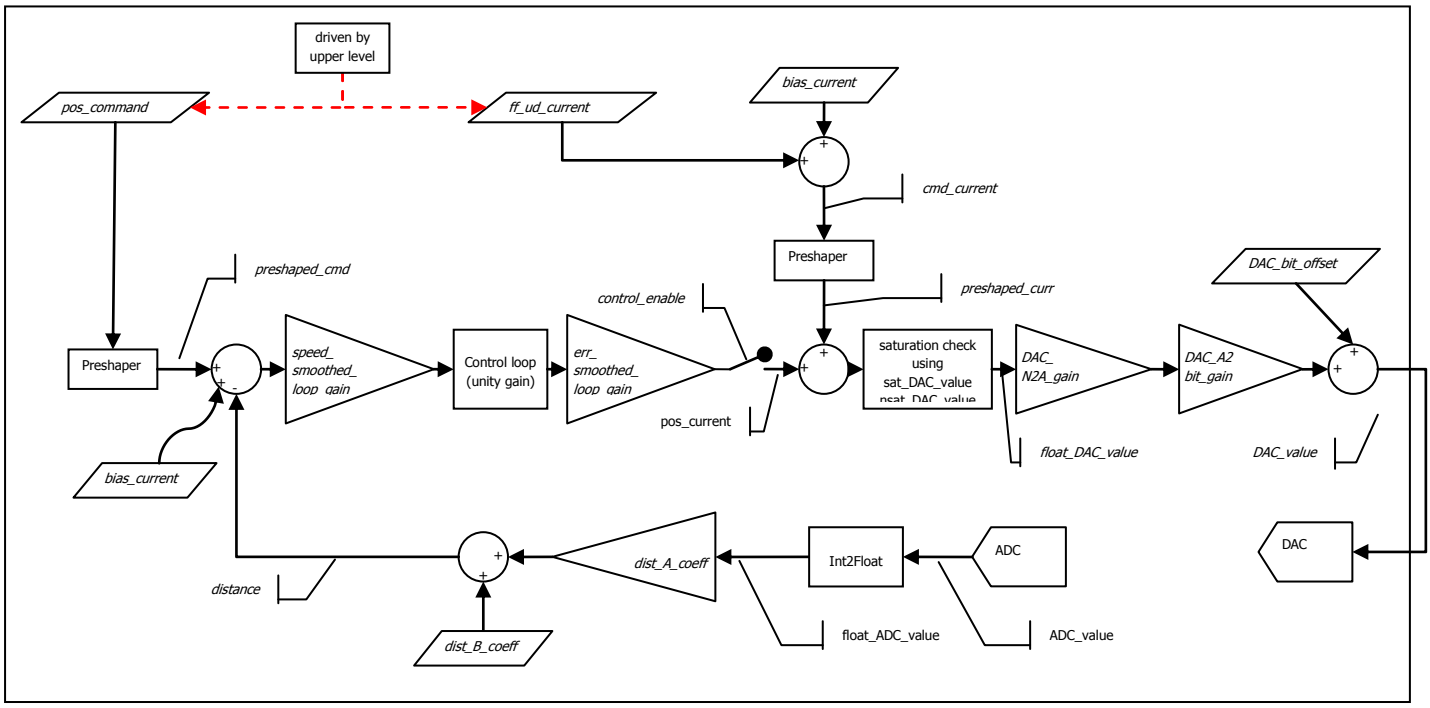

Figure 15 – Actuator local control loop flow chart

The local control loop includes also a mechanism for an accumulation and averaging computation of the commanded actuator voltages and position actuator strain gauges, the mechanism starts at every command update and includes a tunable delay waiting the settling position (i.e. voltage) of the actuator and then a tunable number of accumulated samples. Completed the accumulation, the average is computed and the results are stored to the *_hvcfb_averaged_diagnostic* to be read by the BCU-DSP device for diagnostic purposes, as described in Table 61.

DSP memory area	32 bit dword (float)
_hvcfb_averaged_diagnostic	channel #0 voltage of TT0
_hvcfb_averaged_diagnostic + 1	channel #1 voltage of TT0
_hvcfb_averaged_diagnostic + 2	channel #2 voltage of TT0
_hvcfb_averaged_diagnostic + 3	not used (0)
_hvcfb_averaged_diagnostic + 4	channel #0 voltage of TT1
_hvcfb_averaged_diagnostic + 5	channel #1 voltage of TT1
_hvcfb_averaged_diagnostic + 6	channel #2 voltage of TT1
_hvcfb_averaged_diagnostic + 7	not used (0)
_hvcfb_averaged_diagnostic + 8	channel #0 position of TT0
_hvcfb_averaged_diagnostic + 9	channel #1 position of TT0
_hvcfb_averaged_diagnostic + 10	channel #2 position of TT0
_hvcfb_averaged_diagnostic + 11	not used (0)
_hvcfb_averaged_diagnostic + 12	channel #0 position of TT1
_hvcfb_averaged_diagnostic + 13	channel #1 position of TT1

_hvcfb_averaged_diagnostic + 14	channel #2 position of TT1
_hvcfb_averaged_diagnostic + 15	not used (0)

Table 61 – Averaged actuator voltage and position data vector

8.4.7 HVC dynamic performances

Using the available PI-S-334.1SL mirror integrated with the final mirror, we performed some dynamic tests in order to optimize the local control loop parameters and estimate the performances of the device with respect to the specific use for this application. Consider that the parameter optimization is tuned both on the single step response and command history, in order to obtain the maximum performances for the specific use of the device and also maintaining a sufficient device dynamic stability.

Unfortunately no simulated time history is already available for the final application so we used a tip tilt command history of the E-ELT site atmosphere. Obviously, this time-history doesn't include other contributions like the telescope acceleration noise, but as soon as a it will be available the test can be repeated.

8.4.7.1 Step response

On Figure 16, Figure 17 and Figure 18 we report the result of a simple step response after the parameter optimization.

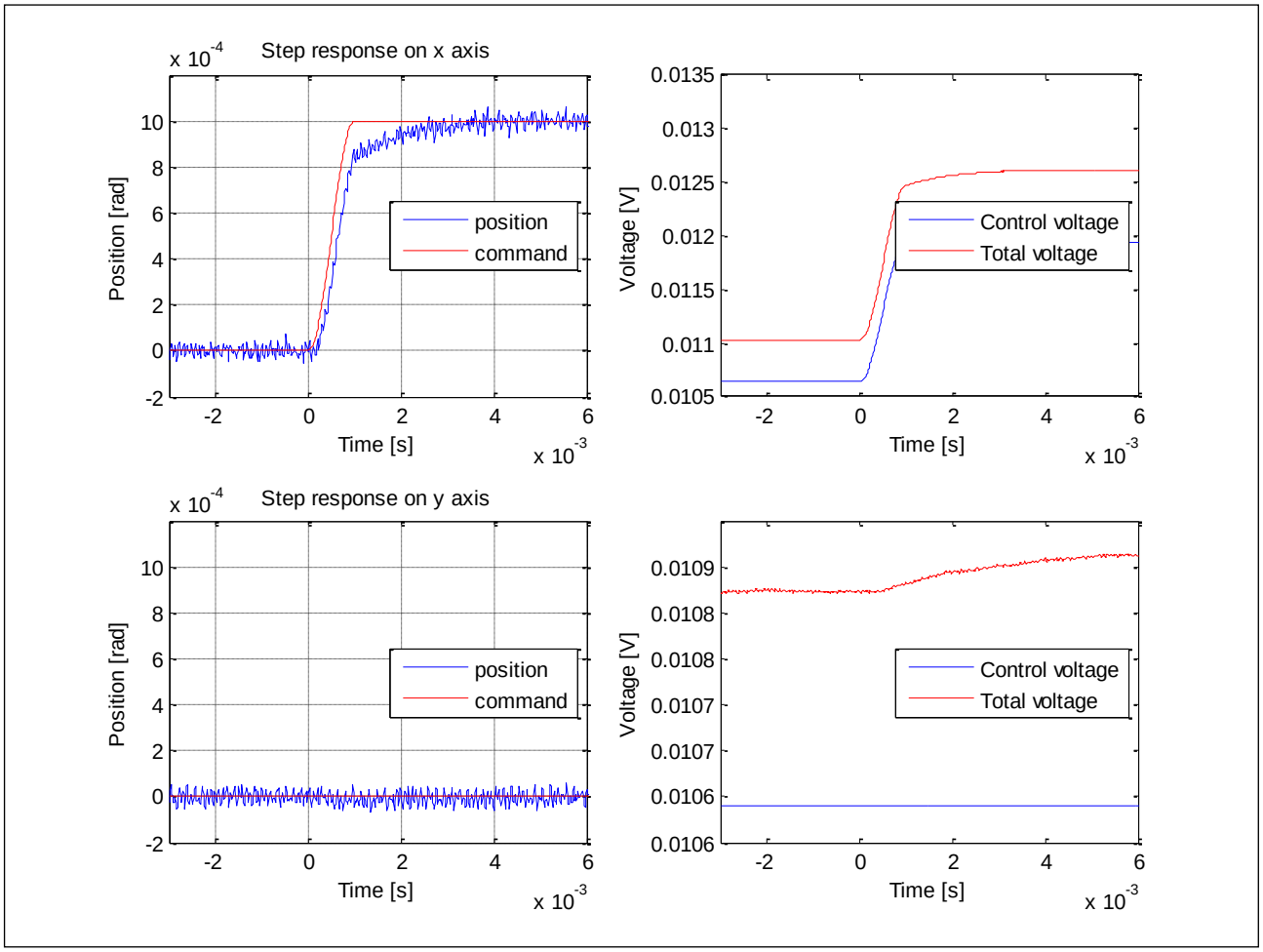

Figure 16 – Step response on x axis

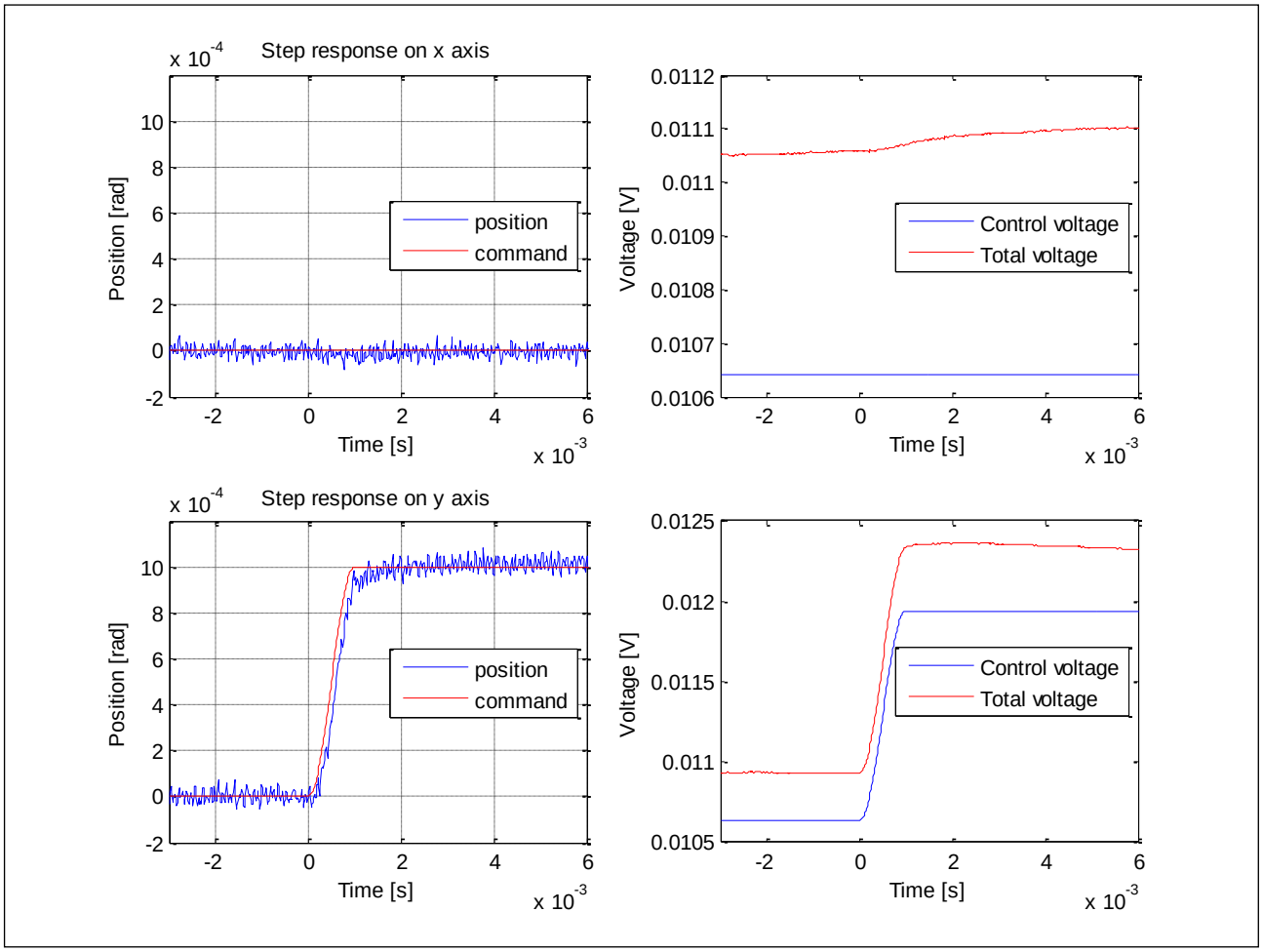

Figure 17 – Step response on y axis

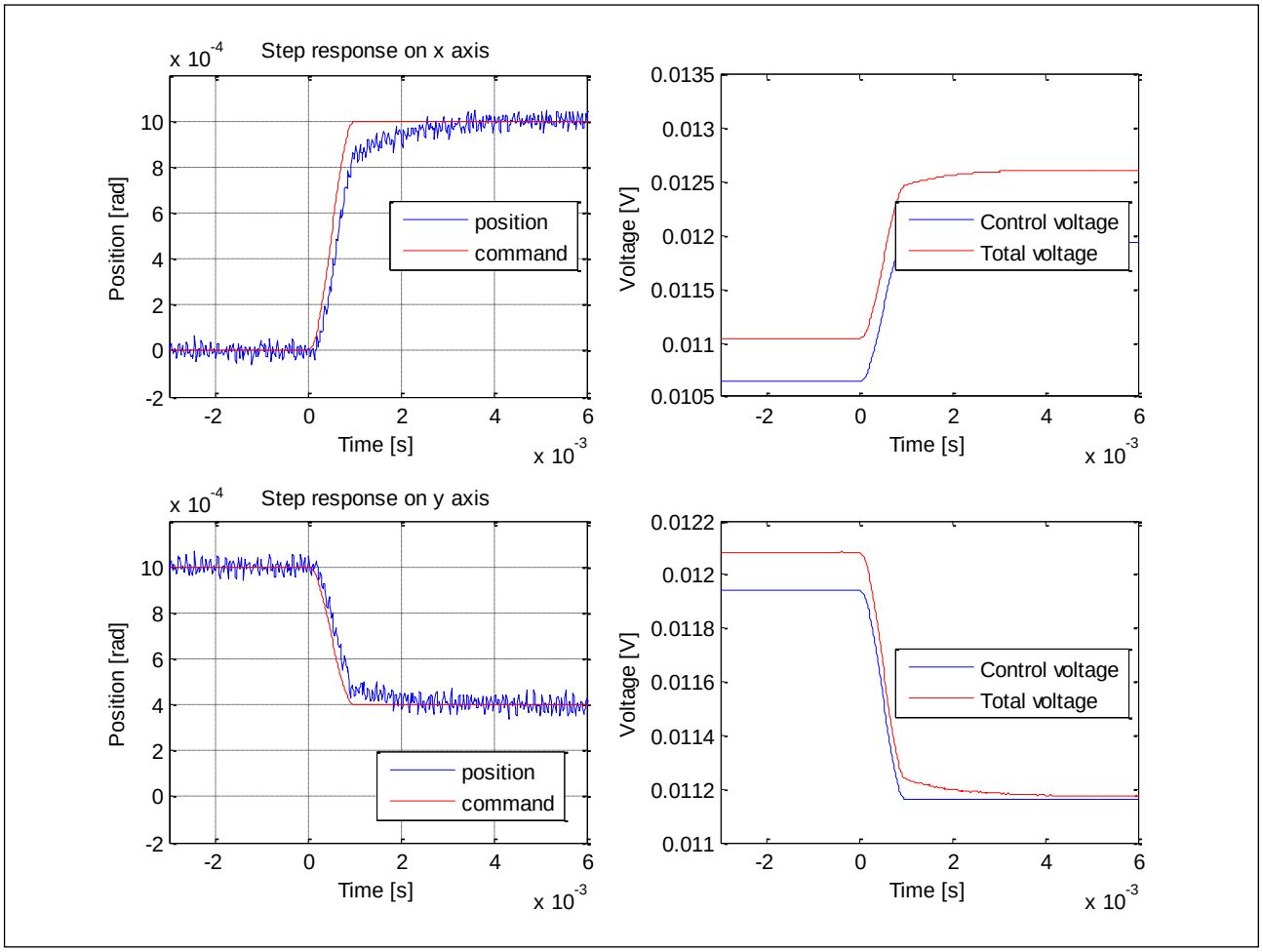

Figure 18 – Step response on both axes

8.4.7.2 Command history response

As mentioned in 8.4.7 the used command history is a simulation of the E-ELT site atmosphere and no other contribution are included, the command history is applied at 1 kHz. The parameter used to estimate the "quality" of the system response is the RMS of the difference between the applied command and the effective mirror position (obtained by the mirror strain gauges).

For the considered command history plotted in Figure 19 we measured a RMS of ~23 µrad.

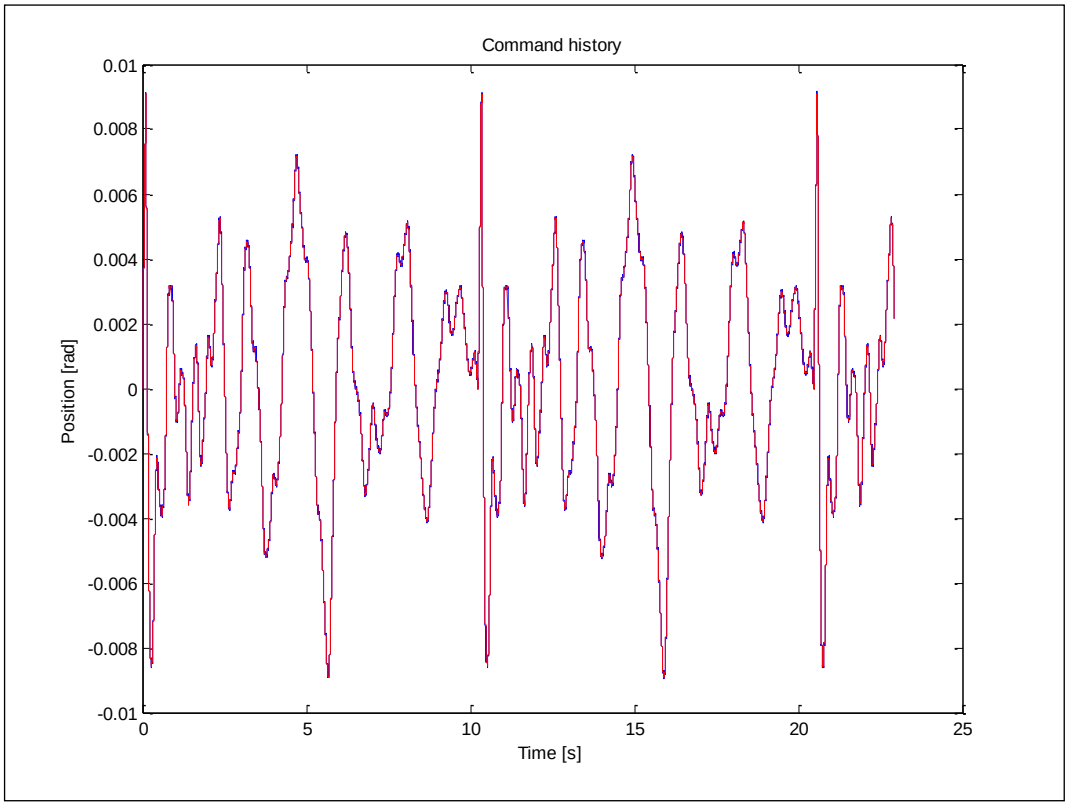

Figure 19 – Mirror response of a command history at 1kHz (x axis)

Figure 20 shows the detail of the behavior of the mirror while it is following the injected commands.

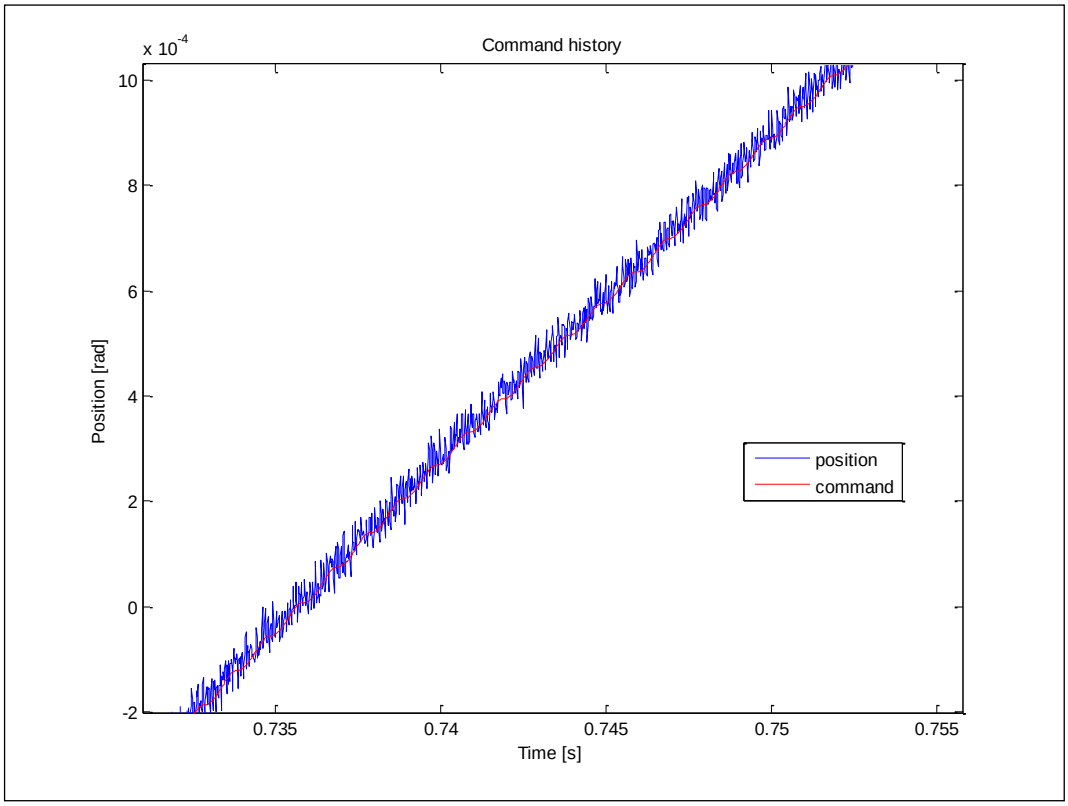

Figure 20 – Detail of the mirror response during the command history

8.4.8 FastLink commands

As FastLink we call all the commands that transfer data from one DSP memory area to another one using the high speed link (at ~4Gbit/s). The use of this kind of commands is related with the real time computation and they should be used with particular attention.

A generic FastLink command uses four register for the initialization as described in Table 62.

Nbits	Bits	of word	Description
8	0-7	0	Reserved char (start of command)
3	8-10	0	Command code: 000 write_same 010 read_sequential
1	11	0	Reserved
1	12	0	Reserved
1	13	0	Generate IRQ on sender DSP device
1	14	0	Generate IRQ on receiver DSP devices
1	15	0	Reserved
16	16-31	0	Data length (number of 32 bit words, 0 to 2047)

12	0-11	1	First DSP (higher 4 bits: crate address; lower 8 bits: DSP address within the crate)
12	12-23	1	Last DSP (higher 4 bits: crate address; lower 8 bits: DSP address within the crate)
8	24-31	1	Command ID
32	0-31	2	Destination DSP start address (in 32 bit addressing mode)
15	0-14	3	Source DSP start address (qword based) without the bank number
2	15-16	3	Source DSP bank number
15	17-31	3	Total data length (in qwords). For the write same command it is equal to data length / 2, for the read sequential command it is equal to data length * number of DSPs involved / 2

Table 62 – FastLink command registers

In the ARGOS BCU mini-crate all the FastLink commands are generated by the BCU-DSP device. All the FastLink command registers used for ARGOS system implementation should be initialized in the *_bscub_FastlinkCmd* memory area. The following Table 63, Table 64, Table 65, Table 66, Table 67 describe the five FastLink commands used in the ARGOS system.

Read sequential of _dscub_SlopeCompleted variable from both DSP devices of DSP-ADC board into _bscub_ReplyVector area			
Memory location	**Dword - Bits**	**Value**	**Description**
_bscub_FastlinkCmd+0	#0: 00 – 07	00 h	For internal use
	08 – 10	010 b	Read sequential command
	11 – 12	00 b	Reserved bits
	13	1 b	Generate IRQ on sender DSP
	14	0 b	Generate IRQ on target DSPs
	15	0 b	Reserved bit
	16 – 31	0002 h	Data length in dword
_bscub_FastlinkCmd+1	#1: 00 – 08	00 h	First DSP of the chain
	09 – 11	0 h	First crate of the chain
	12 – 19	01 h	Last DSP of the chain
	20 – 23	0 h	Last crate of the chain
	24 - 31	0 h	Command ID
_bscub_FastlinkCmd+2	#2: 00 - 31	_dscub_SlopeCompleted	
_bscub_FastlinkCmd+3	#3: 00 – 14	(_bscub_ReplyVector&0x0000FFFF)/2	
	15 – 16	(_bscub_ReplyVector&0x00180000)>>19	
	17 – 31	0004 h	Total data length

Table 63 – FastLink command #0 (this command is not used any more)

Read sequential of _dscfb_SlopeOutput vector from both DSP devices of DSP-ADC board into _bscub_ReplyVector area			
Memory location	**Dword - Bits**	**Value**	**Description**
_bscub_FastlinkCmd+4	#0: 00 – 07	00 h	For internal use
	08 – 10	010 b	Read sequential command
	11 – 12	00 b	Reserved bits
	13	1 b	Generate IRQ on sender DSP

	14	0	b	Generate IRQ on target DSPs
	15	0	b	Reserved bit
	16 – 31	022C	h	Data length in dword
_bscub_FastlinkCmd+5	#1: 00 – 08	00	h	First DSP of the chain
	09 – 11	0	h	First crate of the chain
	12 – 19	01	h	Last DSP of the chain
	20 – 23	0	h	Last crate of the chain
	24 - 31	1	h	Command ID
_bscub_FastlinkCmd+6	#2: 00 - 31	_dscub_SlopeOutputHeder		
_bscub_FastlinkCmd+7	#3: 00 – 14	(_bscub_ReplyVector&0x0000FFFF)/2		
	15 – 16	(_bscub_ReplyVector&0x00180000)>>19		
	17 – 31	0458	h	Total data length

Table 64 – FastLink command #1

Write same to both HVC-DSP devices of _bscfb_TTSlopeVector into _hvcfb_MirrorCommands area				
Memory location	**Dword - Bits**	**Value**		**Description**
_bscub_FastlinkCmd+8	#0: 00 – 07	00	h	For internal use
	08 – 10	000	b	Write same command
	11 – 12	00	b	Reserved bits
	13	1	b	Generate IRQ on sender DSP
	14	0	b	Generate IRQ on target DSPs
	15	0	b	Reserved bit
	16 – 31	8		
_bscub_FastlinkCmd+9	#1: 00 – 08	02	h	First DSP of the chain
	09 – 11	0	h	First crate of the chain
	12 – 19	05	h	Last DSP of the chain
	20 – 23	0	h	Last crate of the chain
	24 - 31	2	h	Command ID
_bscub_FastlinkCmd+10	#2: 00 - 31	_hvcfb_MirrorCommands		
_bscub_FastlinkCmd+11	#3: 00 – 14	(_bscub_TTSlopeVector&0x0000FFFF)/2		
	15 – 16	(_bscub_TTSlopeVector&0x00180000)>>19		
	17 – 31	8		

Table 65 – FastLink command #2

Read sequential from both HVC-DSP devices of _hvcfb_averaged_diagnostic vector into _bscub_ReplyVector area				
Memory location	**Dword - Bits**	**Value**		**Description**
_bscub_FastlinkCmd+12	#0: 00 – 07	00	h	For internal use
	08 – 10	010	b	Read sequential command
	11 – 12	00	b	Reserved bits
	13	1	b	Generate IRQ on sender DSP
	14	0	b	Generate IRQ on target DSPs
	15	0	b	Reserved bit
	16 – 31	0010	h	Data length in dword

_bscub_FastlinkCmd+13	#1:	00 – 08	02 h	First DSP of the chain
		09 – 11	0 h	First crate of the chain
		12 – 19	05 h	Last DSP of the chain
		20 – 23	0 h	Last crate of the chain
		24 - 31	3 h	Command ID
_bscub_FastlinkCmd+14	#2:	00 - 31	_hvcfb_averaged_diagnostic	
_bscub_FastlinkCmd+15	#3:	00 – 14	(_bscub_ReplyVector&0x0000FFFF)/2	
		15 – 16	(_bscub_ReplyVector&0x00180000)>>19	
		17 – 31	0020 h	Total data length

Table 66 – FastLink command #3

Write same to SwtichBCU of DM of _bscfb_SlopeVector + _bscub_StartRTR into swb_SCSlopeVector + swb_SCStartRTR area				
Memory location	**Dword - Bits**		**Value**	**Description**
_bscub_FastlinkCmd+16	#0:	00 – 07	00 h	For internal use
		08 – 10	000 b	Write same command
		11 – 12	00 b	Reserved bits
		13	1 b	Generate IRQ on sender DSP
		14	0 b	Generate IRQ on target DSPs
		15	0 b	Reserved bit
		16 – 31	1604	
_bscub_FastlinkCmd+17	#1:	00 – 08	FF h	First DSP of the chain
		09 – 11	0 h	First crate of the chain
		12 – 19	FF h	Last DSP of the chain
		20 – 23	0 h	Last crate of the chain
		24 - 31	4 h	Command ID
_bscub_FastlinkCmd+18	#2:	00 - 31	swb_SCSlopeVector	
_bscub_FastlinkCmd+19	#3:	00 – 14	(_bscub_SlopeVector&0x0000FFFF)/2	
		15 – 16	(_bscub_SlopeVector&0x00180000)>>19	
		17 – 31	1604	

Table 67 – FastLink command #4

8.4.9 Interface between ARGOS BCU mini-crate and the ARGOS supervisor

The interface between the dedicated electronic for the ARGOS system (i.e. ARGOS BCU mini-crate) and the ARGOS supervisor is based on the Microgate diagnostic communication protocol. This is a dedicated UDP/IP protocol, named MGP (Microgate Protocol).

The structure of the MGP Ethernet packet is presented in Table 68.

Standard	**Name**	**Size (bit)**	**Fixed value**	**Description**
Ethernet	Destination address	48	0x----------	Destination MAC address
Ethernet	Source address	48	0x----------	Source MAC address

Ethernet		Type	16	0x0800		Ethernet protocol used: 0x0800 for IP protocol
Ethernet		Data…				
IP		Version	4	0x4	0x45	IP protocol version: the system uses IPv4
IP		IHL	4	0x5		IP Header length in dwords
IP		Type of service	8	0x00		don't care
IP		Total length	16	0x----		total length of IP&UDP packets in bytes
IP		Identification	16	0x----		the BCU board sets always to 0x0000
IP		Flags	3	0x0000		bit 0: 0 = must be zero bit 1: 1 = don't fragment bit 2: 0 = last fragment
IP		Fragment offset	13			0 = first & last fragment
IP		Time to live	8	0x80		typical value
IP		Protocol	8	0x11		IP protocol used: 0x11 for the UDP protocol
IP		Header checksum	16	0x----		
IP		Source address	32	0x--------		Source IP address
IP		Destination address	32	0x--------		Destination IP address
IP		Data…				
	UDP	Source port	16	0x2710		TCP/UDP port used: 0x2710 for Microgate UDP
	UDP	Destination port	16	0x2710		TCP/UDP port used: 0x2710 for Microgate UDP
	UDP	Length	16			UDP packet length in bytes
	UDP	Checksum	16	0x----		
	UDP	Data…				See the following description
		MGP	dummy word	16	0x0000	
		MGP	header dword #0	32	0x--------	
		MGP	header dword #1	32	0x--------	
		MGP	header dword #2	32	0x--------	
		MGP	Data…			
Ethernet			CRC	32		

Table 68 – Ethernet packet structure

A header and a data payload, depending on the particular command, compose the "data" field of the UDP/MGP protocol.

The command reply is obtained by an asynchronous Ethernet command sent by the BCU board to the host that has issued the command. A counter that is copied on the command reply to associate the command sent with the reply received identifies each command.

Table 69 describes a generic UDP/MGP packet:

	Name	Size (bit)	Description
dummy word		16	Necessary to align the rest of MGP data to 32 bit

header dword #0	first crate	4	don't care
	first DSP	8	first DSP of the crate where the command should be actuated
	last crate	4	don't care
	last DSP	8	last DSP of the crate where the command should be actuated
	command	8	see command table
header dword #1	length	16	packet length in dword
	flags	8	command flags: bit 0 = want reply (to enable the reply packet from communication board to host); bit 1 = as a quad word (to enable burst write transaction writing to DSP).
	command identifier	8	a 8 bit identifier of the packet. The same value is used on reply packet to match the sent command to the reply
header dword #2	start address	32	start address where data are written or read from
Data…	…		if necessary

Table 69 – MGP UDP/IP packet structure

The diagnostic protocol is based on very simple primitives that allow accessing in read and writing all the different memory-mapped devices on BCU, DSP-ADC and HVC boards. The access to these devices is memory mapped as well. Instead of implementing commands, specific memory locations are used as mailboxes to indicate the action to be performed. Parameters can be directly changed by accessing the relevant memory areas.

This method is very flexible and efficient; in fact there is no need, on the ARGOS BCU side, to implement a command parser in strict sense.

This approach has also some drawback, the most evident one being the fact that all high level SW procedures implementing MGP commands need to know the memory mapping of the current firmware running on the target boards. To simplify this potential problem, the spread sheet excel file [RD1] is provided as integral part of the system configuration.

The basic commands available on the MGP protocol are:

- memory write: writes a buffer starting from a specified address to all the specified devices (broadcast);
- memory read: reads a portion of memory starting from a specified address from all specified devices and collect all together.

Beyond these commands there are the following specific commands:

- flash lock: lock a portion of the flash memory to avoid undesired overwriting;
- flash unlock: unlock a portion of the flash memory to allow the overwriting;

- memory clear: clears a portion of the device from a specified address;
- device reset: it is a dedicated command to generate the reset of the specified devices, see 8.4.9.1.

The following table summarizes the memory-mapped devices available on the boards and the corresponding commands. The memory address always starts from zero on each different device.

Board	Device	Command					
		reset	write	read	clear	lock	unlock
BCU	Flash	x	x	x	x	x	x
	Sdram		x	x	x		
	Sram		x	x	x		
	DSP	x	x	x			
DSP-ADC	Flash	x	x	x	x	x	x
	Sdram		x	x	x		
	Sram		x	x	x		
	DSP	x	x	x			
HVC	Flash	x	x	x	x	x	x
	Sdram		x	x	x		
	Sram		x	x	x		
	DSP	x	x	x			

Table 70 – Devices accessible by the diagnostic communication commands

Table 71 reports a complete list of the MGP commands available:

Command name	Value	Description
MGP_OP_WRSAME_DSP	0	Writes the same buffer to the DSPs
MGP_OP_WRSEQ_DSP	1	Writes different buffers to the DSPs
MGP_OP_RDSEQ_DSP	2	Reads a portion of memory from the DSPs
RESERVED CMD	4	This command is not used in the ARGOS system
RESERVED CMD	5	This command is not used in the ARGOS system
RESERVED CMD	6	This command is not used in the ARGOS system
MGP_OP_RESET_DEVICES	10	Resets independently all the devices on the DSM crate
RESERVED CMD	11	This command is not used in the ARGOS system
MGP_OP_LOCK_FLASH	128	Locks a portion of the flash
MGP_OP_UNLOCK_FLASH	129	Unlocks a portion of the flash
MGP_OP_CLEAR_FLASH	130	Clears a portion of the flash
MGP_OP_WRITE_FLASH	131	Writes the same buffer to the flash
MGP_OP_RDSEQ_FLASH	132	Reads a portion of memory from the flash
RESERVED CMD	135	This command is not used in the ARGOS system
RESERVED CMD	136	This command is not used in the ARGOS system
RESERVED CMD	137	This command is not used in the ARGOS system
MGP_OP_CLEAR_SDRAM	140	Clears a portion of the SDRAM
MGP_OP_WRSAME_SDRAM	141	Writes the same buffer to the SDRAM

MGP_OP_WRSEQ_SDRAM	142	Writes different buffers to the SDRAM
MGP_OP_RDSEQ_SDRAM	143	Reads a portion of memory from the SDRAM
MGP_OP_CLEAR_SRAM	145	Clears a portion of the SRAM
MGP_OP_WRSAME_SRAM	146	Writes the same buffer to the SRAM
MGP_OP_WRSEQ_SRAM	147	Writes different buffers to the SRAM
MGP_OP_RDSEQ_SRAM	148	Reads a portion of memory from the SRAM
RESERVED CMD	150	This command is not used in the ARGOS system
RESERVED CMD	151	This command is not used in the ARGOS system
RESERVED CMD	152	This command is not used in the ARGOS system
RESERVED CMD	155	This command is not used in the ARGOS system
RESERVED CMD	156	This command is not used in the ARGOS system
MGP_OP_CMD_SUCCESS	200	Used on the reply packet if the command has been processed with success
MGP_OP_CMD_FAULT	201	Used on the reply packet if a fault condition happened, Typically when the input parameters of the command was wronged
MGP_OP_CMD_TIMEOUT	202	Used on the reply packet if an internal code execution timeout has happened
MGP_OP_CMD_WARNING	203	This command is not used in the ARGOS system
MGP_OP_CMD_DIAGNOSTIC	204	Used on the reply of a diagnostic packet
RESERVED CMD	240	This command is not used in the ARGOS system
RESERVED CMD	241	This command is not used in the ARGOS system
RESERVED CMD	242	This command is not used in the ARGOS system
RESERVED CMD	243	This command is not used in the ARGOS system
MGP_OP_CMD_NULL	255	Null command

Table 71 - MGP commands list

There are also some generic flags used to give some additional information on the command:

Command name	Bit	Description
MGP_FL_WANTREPLY	1	Enable the creation of a reply command to the ARSOG supervisor. To be set also when a read command is sent.
MGP_FL_ASQUADWORD	2	Used in the write to DSP memory commands. It enables a special write mode without break during the data transfer
RESERVED FLAG	3	This flag is not used in the ARGOS system
RESERVED FLAG	4	This flag is not used in the ARGOS system
RESERVED FLAG	5	This flag is not used in the ARGOS system

Table 72 - MGP flags list

Hereafter we report an example of a MGP command:

MGP_OP_WRSAME_DSP		this command allows to write the same buffer to all the DSPs as defined into the header	
	Name	**Size (bit)**	**Value**
header	first crate	4	0x00

dword #0	first DSP	8	0x00 - 0xFF
	last crate	4	0x00
	last DSP	8	0x00 - 0xFF
	command	8	MGP_OP_WRSAME_DSP
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY MGP_FL_ASQUADWORD
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to write the data
Data…		length	data buffer
Reply command if requested			
	Name	**Size (bit)**	**Value**
header dword #0	first crate	4	0x00
	first DSP	8	0x00
	last crate	4	0x00
	last DSP	8	0x00
	reply_command	8	possible commands reply: MGP_OP_CMD_SUCCESS MGP_OP_CMD_FAULT MGP_OP_CMD_TIMEOUT
header dword #1	reply_length	16	0x0000
	flags	8	0x00
	command identifier	8	same command identifier of the master command
header dword #2	start address	32	0x00000000
Data…		0	not used

Table 73 - MGP_OP_WRSAME_DSP command

8.4.9.1 Ethernet “reset” command

The “reset” command (where “reset” is not the most suitable name, given the various functions linked to this command) is a dedicated Ethernet command used to act directly to some board devices respect with the standard Ethernet command that are typically based accessing in read and write all the different memory-mapped devices.

The Table 74 describes the command protocol and the Table 75 lists all the available “reset” commands respect with the board. For each command there is also the link to the use of the bit.

MGP_OP_ RESET_DEVICES		this command resets all the devices specified on the command header	
	Name	**Size (bit)**	**Value**
header dword #0	first crate	4	0x00
	first DSP	8	0x00 - 0xFF

	last crate	4	0x00
	last DSP	8	0x00 - 0xFF
	Command	8	MGP_OP_RESET_DEVICES
header dword #1	Length	16	0x0001
	Flags	8	0x00
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	0x00000000
Data…		2*32	see the following table for the type of reset available
Reply command if requested			
	Name	**Size (bit)**	**Value**
header dword #0	first crate	4	0x00
	first DSP	8	0x00
	last crate	4	0x00
	last DSP	8	0x00
	Command	8	MGP_OP_CMD_SUCCESS or MGP_OP_CMD_FAULT
header dword #1	Length	16	0x0000
	Flags	8	0x00
	command identifier	8	same command identifier of the master command
header dword #2	start address	32	0x00000000
Data…		0	not used

Table 74 – MGP_OP_RESET_DEVICES command

bit	BCU board	HVC board	DSP board
		First DWORD	
0	reset all crate (conf A)	not used	not used
1		not used	not used
2	reset all boards excluded the communication board (conf A)	not used	not used
3		not used	not used
4	not used	reset board	reset board
5	not used		
6	reset FPGA (conf A)	reset FPGA	reset FPGA
7			
8	reset FLASH (conf A)	reset FLASH	reset FLASH
9			
10	reset DSP (conf A)	reset DSP #0	reset DSP #0
11			
12	not used	not used	reset DSP #1
13	not used	not used	
14	reset Ethernet chip (conf A)	not used	not used
15		not used	not used
16	analog supply control (conf B)	not used	not used
17		not used	not used

18	coil power supply control (conf B)	not used	not used
19		not used	not used
20	Ethernet watchdog control (conf C)	not used	DSP watchdog control (conf B)
21		not used	
22	reserved for internal use	not used	not used
23	reserved for internal use	not used	not used
24	reset of timestamp global register	not used	not used
25	on boards led (conf B)	on boards led (conf B)	on boards led (conf B)
26			
27	input FastLink port enabling/disabling (conf E)	not used	not used
28	not used	not used	not used
29	not used	not used	not used
30	not used	not used	not used
31	not used	not used	not used
Second DWORD			
0	current threshold volatile registers update (conf D)	not used	coil driver #0 control (conf B)
1		not used	
2		not used	coil driver #1 control (conf B)
3	set the power backplane serial number (conf E)	not used	
4	set the frequency of the signal reference signal (conf E)	not used	coil driver #2 control (conf B)
5	current threshold non-volatile registers update (conf D)	not used	
6		not used	coil driver #3 control (conf B)
7		not used	
8	not used	not used	coil driver #4 control (conf B)
9	not used	not used	
10	not used	not used	coil driver #5 control (conf B)
11	not used	not used	
12	not used	not used	coil driver #6 control (conf B)
13	not used	not used	
14	not used	not used	coil driver #7 control (conf B)
15	not used	not used	
16	Argument of command	not used	not used
17		not used	not used
18		not used	not used
19		not used	not used
20		not used	not used
21		not used	not used
22		not used	not used
23		not used	not used
24		not used	not used
25		not used	not used
26		not used	not used
27		not used	not used
28		not used	not used
29		not used	not used
30		not used	not used

31		not used	not used

Table 75 – Argument of MGP_OP_RESET_DEVICES command

Bit configuration A	**Operations available**
00	do nothing
01	de-assert the signal reset
10	generate a sequence of reset if the device was operating
11	assert the signal reset
Bit configuration B	**Operations available**
00	do nothing
01	disable
10	enable
11	do nothing
Bit configuration C	**Operations available**
00	do nothing
01	disable
10	enable
11	refresh watchdog
Bit configuration D	**Operations available**
000	do nothing
001	set the value on channel 1
010	set the value on channel 2
011	set the value on channel 3
100	set the value on channel 4
111 - 101	do nothing
bits 16 - 31	data argument of the command
Bit configuration E	**Operations available**
0	do nothing
1	set the value defined in the data argument word (bits 16 – 31)
bits 16 - 31	data argument of the command

Table 76 – List and description of the available bit configurations

8.4.9.2 Ethernet command to enable/disable input FastLink interfaces

A dedicated command has been introduced to enable/disable the input FastLink interfaces: pnCCD and FLAO/Na. At startup and after a system reset both input ports are disabled so any input frame is discarded. The enable/disable command is based on the MGP_OP_RESET_DEVICES command setting to one the bit #27 and writing the proper value to the data word according to the Table 77 (see also Table 75).

bit	**First DWORD**	**Second DWORD - Command Data Word (MSW)**
27	bit to enable/disable FastLink ports	• 0 - disable all ports • 1 - enable pnCCD port • 2 - enable FLAO / Na BCU port

Table 77 – FastLink input port selection

This original command has been modified after the BCU logic version 11.70, in fact the pnCCD optical interface has been removed and then the mentioned enabling command is not relevant any more. In any case the FLAO / Na BCU port should be still enabled when it is used and the new command data is 3 instead of 2, Table 78 is the updated table to use when the BCU logic version 11.70 or higher it is used.

bit	First DWORD	Second DWORD - Command Data Word (MSW)
27	bit to enable/disable FastLink ports	• 0 - disable all ports • 3 - enable FLAO / Na BCU port

Table 78 – Updated fastLink input port selection for BCU logic version 11.70 or higher

8.4.9.3 Ethernet command to enable/disable pnCCD direct analog input interfaces

Also for the pnCCD direct analog input an enable/disable command is available.
The command is based on the MGP_OP_RESET_DEVICES command but should be addressed to the DSP-ADC board instead to the BCU board. See 9.2 for the command description.

9 Simulating pnCCD frames sequence

9.1 pnCCD frames manual simulation

As requested in [AD1] a debugging mechanism to check the slope computation and data flux in the BCU ARGOS system has been implemented. The basic idea is to write directly to the DSP devices memory of the DSP-ADC board the pnCCD and, writing in the computational trigger registers, move forward the slope computation. In Table 79 there is the command sequence list. All the '*_dscxx_*' variables should be written to both DSP devices of DSP-ADC board.

Command #	DSP memory area	Data value
0	_dscub_NumLinesToDo	write 0 to reset the number of lines to compute
	_dscub_FramesCounter	increment of 1 respect to the previous value starting from 0, this action reset the computational machine
1	_dscub_PixelArea	frame header doubled as indicated in Table 17 + pixels of first line of all CAMEX in the right order
2	_dscub_NumLinesToDo	write 1 (number of lines downloaded)
After this register write command the first line (all CAMEX) is processed and the partial computation can be verified reading the DSP memory		
3	_dscub_PixelArea + 8 + 528/2 (uint16)	pixels of second line of all CAMEX in the right order
4	_dscub_NumLinesToDo	2 (number of lines downloaded)
After this register write command the second line (all CAMEX) is processed and the partial computation can be verified reading the DSP memory		
……		
repeat for all the 132 CAMEX lines		
……		
264	_dscub_PixelArea + 8 + 131 x 528/2 (uint16)	pixels of second line of all CAMEX in the right order
265	_dscub_NumLinesToDo	132 (number of lines downloaded)
This two writes complete the pixels download and automatically the DSP firmware completes the algorithm that can be check verifying if the _dscub_SlopeCompleted is set to 1. If enabled also the pixel frame is stored to the SDRAM memory.		
At this point is possible return to the command #0 and repeat the computation with a new frame but is also possible to proceed with the algorithm sequence…		
0	_bscub_ResetLoop	1 (trigger value to start the BCU –DSP activity)
Now the entire slope algorithm is completed, including the mirrors update (if enabled) and the diagnostic storage and download to the ARGOS supervisor (if enabled). Returning to the command #0 is possible restart with a new frame.		

Table 79 – Command sequence for pnCCD emulation procedure

9.2 Internal generator of pnCCD frames

Introducing the new DSP-ADC 8ch board, we implemented in the FPGA of the DSP board itself an internal frame generator useful both for a step by step frame simulation and for the real time system debugging.

The internal frame generator emulates the ADC reading with the same timing of the sequencer board, physically reading also the ADCs and writing the read data to the DSP memory and then emulating the entire computational mechanism. For these reasons the internal frame generator allows debugging the system initialization, the code computation and the diagnostic storage.

The interface includes also a command to enable/disable the external pnCCD input interfaces. When the internal frame generator is used first the external interface should be disabled. Note that when the enable interface command is sent, the interface is enabled at the first subsequent start of frame and not immediately with the risk that spurious pixels arrives to the DSP devices.

The use of the internal generator of pnCCD frames is controlled by the following commands:

Command #	Data value
0x00000001	disables the internal frame generator (this is the default condition upon system reset)
0x00000002	enables the internal frame generator
0x00000004	releases the signal that generates the single frame, to be ready to send another frame (see above)
0x00000008	generates a single frame (one shot)
0x00000010	disables continuous frame streaming
0x00000020	enables the continuous frame generation at the fixed frequency of 1kHz
0x00000040	de-assert the output signal to disable the Sequencer board
0x00000080	assert the output signal to enable the Sequencer board
0x00000100	disables the external pnCCD analog inputs
0x00000200	enables the external pnCCD analog inputs

Table 80 – Availables commands to control the pnCCD internal generator

Each command should be sent to the DSP-ADC board using the primitive MGP_OP_RESET_DEVICES according to the following format: [0 command 0].

As an example the correct format using the Matlab library is the following:

```
mgp_op_reset_devices(0,0,[0 hex2dec('00000001') 0]); % disables internal trigger
mgp_op_reset_devices(0,0,[0 hex2dec('00000002') 0]); % enables internal trigger
mgp_op_reset_devices(0,0,[0 hex2dec('00000004') 0]); % releases signal to generate a single frame
mgp_op_reset_devices(0,0,[0 hex2dec('00000008') 0]); % generates a single frame
mgp_op_reset_devices(0,0,[0 hex2dec('00000010') 0]); % disables continuous frame generation
mgp_op_reset_devices(0,0,[0 hex2dec('00000020') 0]); % enables continuous frame generation
mgp_op_reset_devices(0,0,[0 hex2dec('00000100') 0]); % disables the external pnCCD analog inputs
mgp_op_reset_devices(0,0,[0 hex2dec('00000200') 0]); % enables the external pnCCD analog inputs
```

10 Real-time computation timing analysis

This chapter reports the timing analysis of the real time computation.

The system is configured as follow:

- the pnCCD input frame has been simulated using the frame generator (see [AD2]) configured to generate a CCD of 264x264 pixels at ~1kHz with:
  - 8.152 µs from the start of frame and the first pixel download
  - 0.593 µs to download one line
  - 1.772 µs of line delay between each line
  - 71.936 µs of final delay before sending the following SOF
  - then the total frame period is: 8.152 + 0.593 * 132 + 1.772 * 131 + 71.936 = 1024.096 µs

The following logic analyzer screen shots show the frame download activity and the detail of the computation and data transfer activity.

In particular:

- D0 is the start of frame pulse at the beginning of the frame
- D1 is the pixel clock triggering the pixel download
- D2 is the DSPs of DSP-ADC board computational phase (active when low)
- D3 is the signal notification to the BCU that a new set of slopes are ready to be sent and the secondary mirror (active when low)
- D4 pixels data transfer from the DSPs memory to the DSP-ADC board SDRAM memory
- D5 is the DSP of BCU board computational phase (active when low)
- D6 bus data transfer activity (active when low)
- D7 pixels and slopes diagnostic buffers transfer to the BCU board SDRAM memory from either DSP device or from DSP-ADC board SDRAM memory.

The Figure 21 shows a typical frame step, and the Figure 22 shows the detail of the last part of the computational lag:

- at 0ps the trigger detects a new SOF (at the rising edge)
- as soon as the first pixel is arrived the slope computation starts (see falling edge of D2 trace)
- the computation is long as the pixels download plus a final computation phase to complete the last slopes and common computation that can be done once all the slopes are ready (see raising edge of D2 trace)

- this time strongly depends to the pnCCD configuration in particular to the pixel download rate, to the slope pixels position and finally to the best computation flux optimization (see 8.1 and 8.2) then the measured time of 1.0046 ms of the following plot is just an example and not representative of a possible real case.
- more deterministic is the subsequent steps that are not affected to the pnCCD configuration, at the rising edge of D2, the end of computation signal is asserted (see falling edge of D3) to notify to the DSP device of the BCU board that a new set of slopes is ready;
- as soon as the signal is triggered the new set of slopes in transferred from the DSP-ADC board to the BCU board, (lower phase of D6 trace)
- then the DSP device of BCU board completes the slope computation collecting the slopes of the two DSPs of the DSP-ADC board, the FLAO slopes and the TT slopes, and finally sends the new slope vector to the SwitchBCU of the secondary mirror; as indicated the duration of this phase is about 76.9 µs (see lower phase of D5 trace)
- looking on Figure 22 we have a more accurate measurement of the delay introduced by this final part of the computation with respect to the end of the slope computation executed on the DSP-ADC board and that, as above mentioned, depends to the pnCCD configuration. This delay is 95.0 µs and at the rising edge of the D5 trace the new slope vector has arrived to the secondary mirror.
- regarding the diagnostic storage, from Figure 21 we measured 595.1 µs for the data transfer of one entire pnCCD frame (of 1.063 Mbit as described in 8.4.1) from the DSP memory to the on board SDRAM memory of the DSP-ADC board (see longer low phase of the D4 trace). The D7 trace triggers the diagnostic storage on the BCU board, the first shorter higher phase of the D7 trace (on the left) represent the slope buffer (of 52.992 Kbit as described in 8.4.2) storage to the SDRAM memory, the measurement of this slot of 36.2 µs is indicated in Figure 22. The longer higher phase of D7 trace triggers the storage of both slopes and pnCCD pixels frame. This high phase includes at the beginning the slope diagnostic buffer storage, then an hold phase waiting that the transfer on the SDRAM of the DSP-ADC board is completed (about 700 µs) and then the transfer of the pnCCD pixels frame from the DSP-ADC board to the BCU board. The measured total time is 1.3384 µs. To be notice that the subsequent slope diagnostic buffer storage is delayed but well managed and it is executed as soon as the previous large data transfer is completed.
- analyzing Figure 21 we could assume that the maximum acceptable diagnostic storage frequency is all frames for the slope diagnostic buffer and half frames for the pnCCD pixels frames. This consideration is valid only if the diagnostic is stored on the SDRAM memory and the masterBCU mechanism is disabled (see 8.4.3.2). In fact, if this useful mechanism is enabled, the real bottleneck is the actual Ethernet bandwidth, estimated to be 100Mbit/s, as described on 8.4.2, therefore significantly slower than the theoretical bandwidth of 1Gbit/sec.

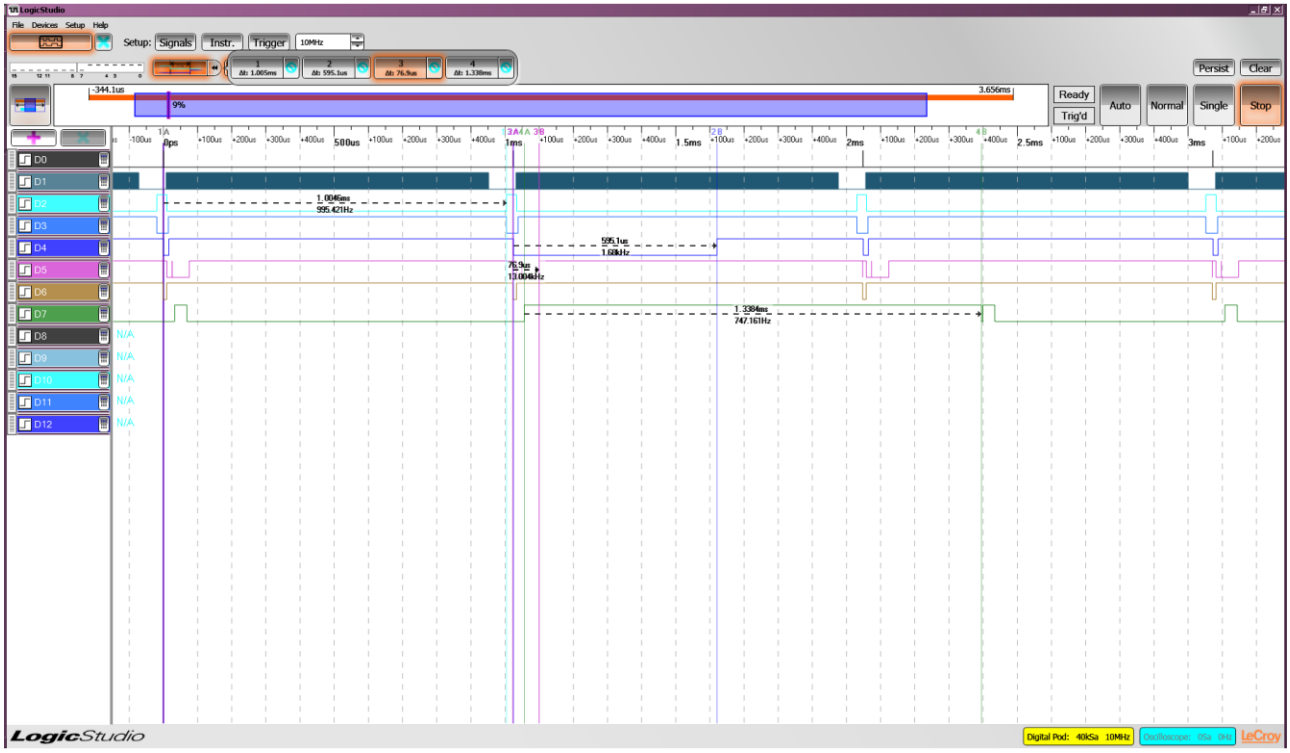

Figure 21 – Sequence of three pixel frames at 1kHz of frequency

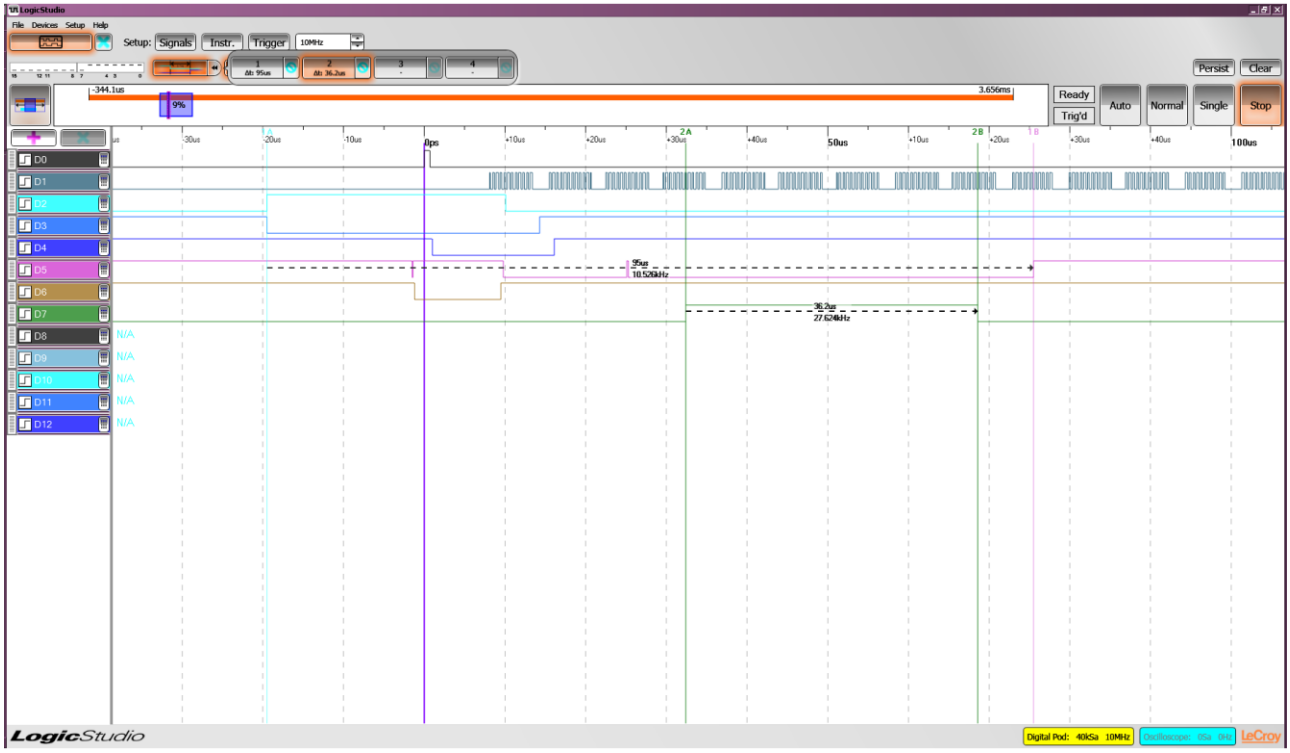

Figure 22 – Detail of the last phase of slope computation once that the last pixel is arrived

11 Conclusion

The document provides the detailed design description of the ARGOS BCU system. Beyond the design aspects, it also provides a guideline for the proper initialization of the system with a detailed description of all involved parameters.

12 ANNEXES

12.1 Connector pin-out for MPIfR serial interface

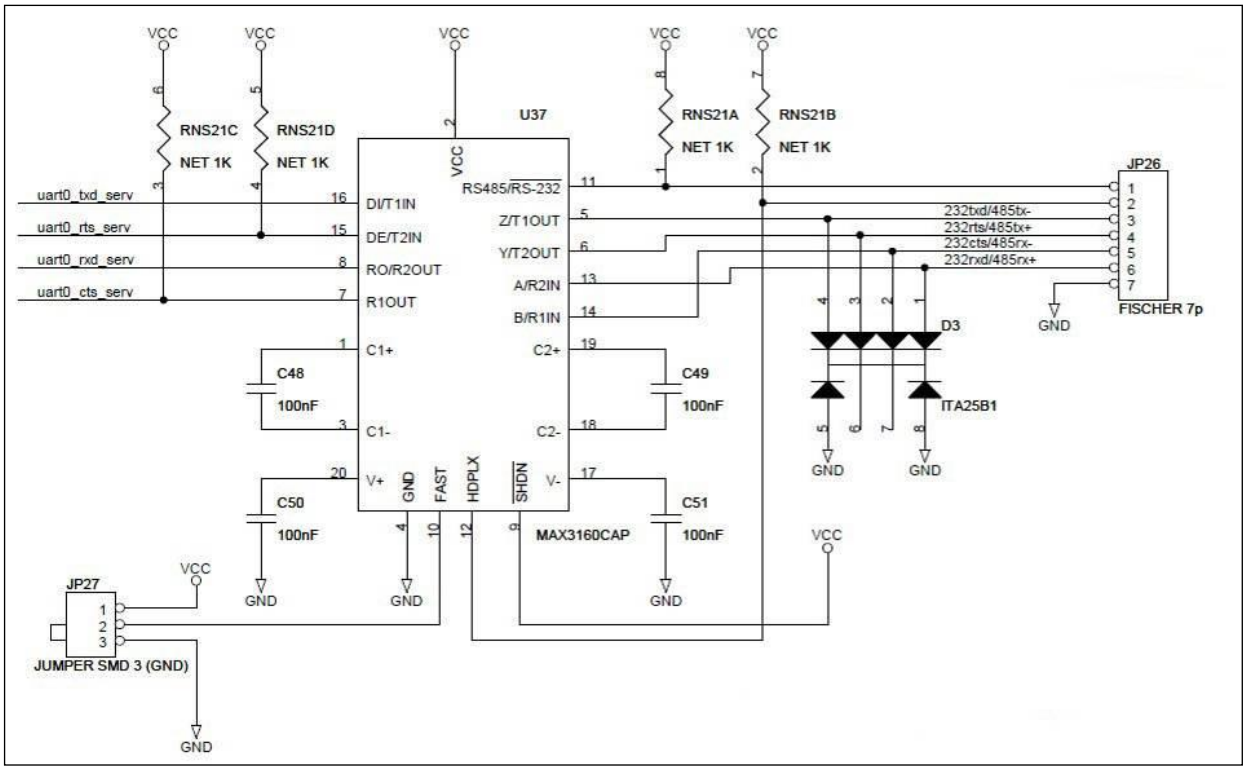

Figure 23 – RS232/485 Serial connector pin-out