MICROGATE

KECK NEXT GENERATION WAVEFRONT CONTROLLER

Real Time Controller
Detailed Design Review Data Package

Document :	NGWFC_RTC_DDR_001.doc	
Issue :	1	
Date :	December 2nd, 2005	
Prepared by :	MICROGATE	
	R.Biasi	.....
	D.Pescoller	.....
	M.Andrighettoni	.....
Checked by :		.....
Approved by :		.....
Released by :		.....

December 2nd, 2005

CHANGE RECORDS

ISSUE	DATE	Author	Approved	QA/ QC	SECTION / PARAG. AFFECTED	REASON/INITIATION DOCUMENTS/REMARKS
1	12.02.2005	Microgate			All	First Issue

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 ACRONYMS

AO Adaptive Optics
CCD Charge Coupled Device
CIE Command Interpreter and Executer
COTS Commercial Off-The-Shelf
DDR Double Data Rate
DM Deformable Mirror
DMA Direct Memory Access
DSP Digital Signal Processor
DTT Down Tip Tilt
DTTM Down Tip Tilt Mirror
FC-IP FibreChannel Internet Protocol
FITs Number of Failures in 10^9 hours
FPDP Front Panel Data Port
GPIB General Purpose Interface Bus
HBA Host Adapter Board
HP Width unit for 19" chassis, corresponding to 0.2" (5.08mm)
HV High Voltage
HVA High Voltage Amplifier
HVC High Voltage Control
ICMP Internet Control Message Protocol
IIR Infinite Impulse Response
LFpM Linear Feet per Minute
LAN Local Area Network
LGS Laser Guide Star
LUT Look Up Table
MAC Multiply And Accumulate
mas milliarcseconds
MGAOS Microgate Adaptive Optics real-time System
MIMO Multiple Input Multiple Output
MIL-STD military standard
MMF Multi-Mode Fiber
NDA Non Disclosure Agreement
NFS Network File System
NGS Natural Guide Star
NGWFC Next Generation Wavefront Controller
PCB Printed Circuit Board
PIO Programmable Input Output

PSU	Power Supply Unit
RMS	Root-Mean-Square
RTC	Real Time Controller
SAN	Storage Area Network
SAS	Serial Attached SCSI
SCSI	Small Computer System Interface
SFP	Small Form factor Pluggable
SI	The International System of Units
SH	Shack-Hartmann
SRAM	Static Random Access Memory
SDRAM	Synchronous Dynamic Random Access Memory
STRAP	System for Tip-tilt Removal with Avalanche Photo-diodes
TBC	To Be Confirmed
TBD	To Be Defined
TRS	Telemetry Recorder/Server
U	Height unit for 19” chassis, corresponding to 1.75” (44.45mm)
UTT	Uplink Tip Tilt
UTTM	Uplink Tip Tilt Mirror
VME	VersaModule Eurocard
WBS	Work Breakdown Structure
WCP	Wavefront Controller Command Processor
WIF	Wavefront Controller Interface
WFP	Wavefront processor
WFS	Wavefront Sensor

2 APPLICABLE DOCUMENTS

[AD1] CARA/W.M. Keck
NGWFC RTC Requirements – Keck Adaptive optics note #311. Version 1.0, March 11th, 2005

[AD2] CARA/W.M. Keck
NGWFC RTC Tip-Tilt Requirements – Keck Adaptive optics note #329. Version 1.0, May 25th, 2005

[AD3] CARA/W.M. Keck
NGWFC RTC Vendor Statement of Work – Keck Adaptive optics note #310. Version 1.0, March 11th, 2005

[AD4] CARA/W.M. Keck
NGWFC System Design Manual – Keck Adaptive optics note #289. Version 2.0, August 15th, 2005

[AD5] Microgate S.r.l.
Real Time Controller Preliminary Design Review Data Package
Issue 1 – August 22nd, 2005

[AD6] CARA/W.M. Keck
Request for change to the NGWFC RTC: Post PDR updates - Keck Adaptive optics note #354. Version 1.4, Novemeber 3rd, 2005

[AD7] CARA/W.M. Keck
NGWFC RTC Acceptance Test Plan - Keck Adaptive optics note #374

[AD8] CARA/W.M. Keck
NGWFC Detailed Design Report - Keck Adaptive optics note #371
December 2nd, 2005

3 REFERENCE DOCUMENTS

[RD1] R. Biasi, M.Andrighettoni et al. - 'Dedicated flexible electronics for adaptive secondary control', - SPIE Proc. on 'Advancements in Adaptive Optics', 5490, p.1502

[RD2] E-mails exchanged between Microgate and CARA-Keck between April 21st ad May 7th, 2005

[RD3] Department of Defense USA, MIL-HDBK-217 Revision F, Reliability Prediction of Electronic Equipment

[RD4] Keck AO Wavefront Control –Hardware Manual

[RD5] INCITS - FibreChannel – Physical and Signaling Interface – ANSI – INCITS 230-1994

[RD6] M. Rajagopal, R. Bhagwat, W. Rickard - RFC 2625 - IP and ARP over FibreChannel - June 1999

4 INTRODUCTION

This document contains the Detailed Design Review data package of those parts of the NGFWC Real Time Controller that are under responsibility of Microgate.

In this perspective, it can be considered as a deeper description of the topics already outlined in [AD8].

The document derives directly from the preliminary design review report ([AD5]).

Even if the entire document has been thoroughly reviewed, addressing in particular performance analysis and design details, we report hereafter a list of the major additions and modifications with respect to the original document.

- Assessment of HVC analog simulation
- Results of HVC analog circuit breadboarding
- WIF-WCP interface
- Command Interpreter and Executer
- TRS database (here we introduced a major change with respect to PDR)
- Results of preliminary tests on TRS performance
- Validation of real-time computation performance estimate through testing of algorithms and C versus assembler comparison
- Assessment of interface test between RTC and TRS including test of IP over FibreChannel communication

An ’Annex’ section has been added at the end of the document, including:

- Real Time Code examples
- Low level MGAOS/MVME communication protocol (MGP)
- WIF-WCP interface command list
- Final choice of COTS components (jointly with CARA)
- Schematics of all proprietary boards, including component placement and bill of material (these are reported as links to external documents)

The Verification matrix has been removed from the document, because it is already present in other design documents ([AD8], [AD7]).

Moreover, we also removed the section dedicated to acceptance test at Microgate premises because this topic is specifically addressed by [AD7]

5 SYSTEM OVERVIEW

In this section we present a general overview of the preliminary NGWFC RTC design.

In Figure 1 we present a global block diagram of the designed architecture.

The main system functional blocks of NGWFC RTC are:

- WaveFront Processor (WFP): it is the hearth of the wavefront controller, performing all real-time computations from pixel acquisition to generation of commands for the DM and DTT/UTT mirrors.
- Wavefront controller Interface (WIF): this is the interface between external commands (from WCP) to the WFP.
- DTT/UTT controllers, implementing closed loop control of the piezoelectric-actuated DTT and UTT mirrors.
- STRAP tip-tilt sensor and controller.
- Telemetry Recorder Server (TRS): it is the main diagnostic storage system, receiving streams of data from WFP, STRAP, WIF and WCP.

All this components can be clearly distinguished in the block diagram of Figure 1, describing also the hardware devices and interfaces where the functions above are actually implemented.

The ***WFP*** is implemented by the Microgate Adaptive Optics real-time System (MGAOS). This is a proprietary hardware architecture that was developed in the frame of other adaptive optics projects, with the aim of providing a well-optimized and scalable high performance computational and control architecture. All MGAOS components are mounted on a single proprietary bus and the whole system is fit into a small case integrated in the VME crate (see Figure 2). The MGAOS is directly interfaced to the Wavefront Sensor Camera (standard AIA interface) and to the DM HVA (FPDP-like interface). The ***DTT/UTT controllers*** are also integrated in the MGAOS, thus a direct interface between MGAOS and DTT/UTT mirrors is provided.

The ***WIF*** is implemented on a standard VME architecture (PowerPC VME 6100 CPU board). The WIF is the interface between WCP and WFP. It also acts as supervisor of all RTC operations. In particular, all setup and configuration of MGAOS are performed by the VME CPU through a private Gbit Ethernet connection. The same interface allows also transferring real-time data from ***STRAP*** to MGAOS.

The ***TRS*** comprehends a storage data server and a large disk array for data storage, capable of storing the system telemetry for longer than five observing nights. It is interfaced to MGAOS through a dedicated FibreChannel link, through which all bulky telemetry data are transferred. STRAP and configuration diagnostics is transferred directly from the VME CPU via a standard Gbit Ethernet LAN. The client workstation can query the TRS by means of the same standard LAN.

An IRIG board installed on the VME crate provides system synchronization to an absolute timing reference.

An external reset line allows to reset the RTC remotely.

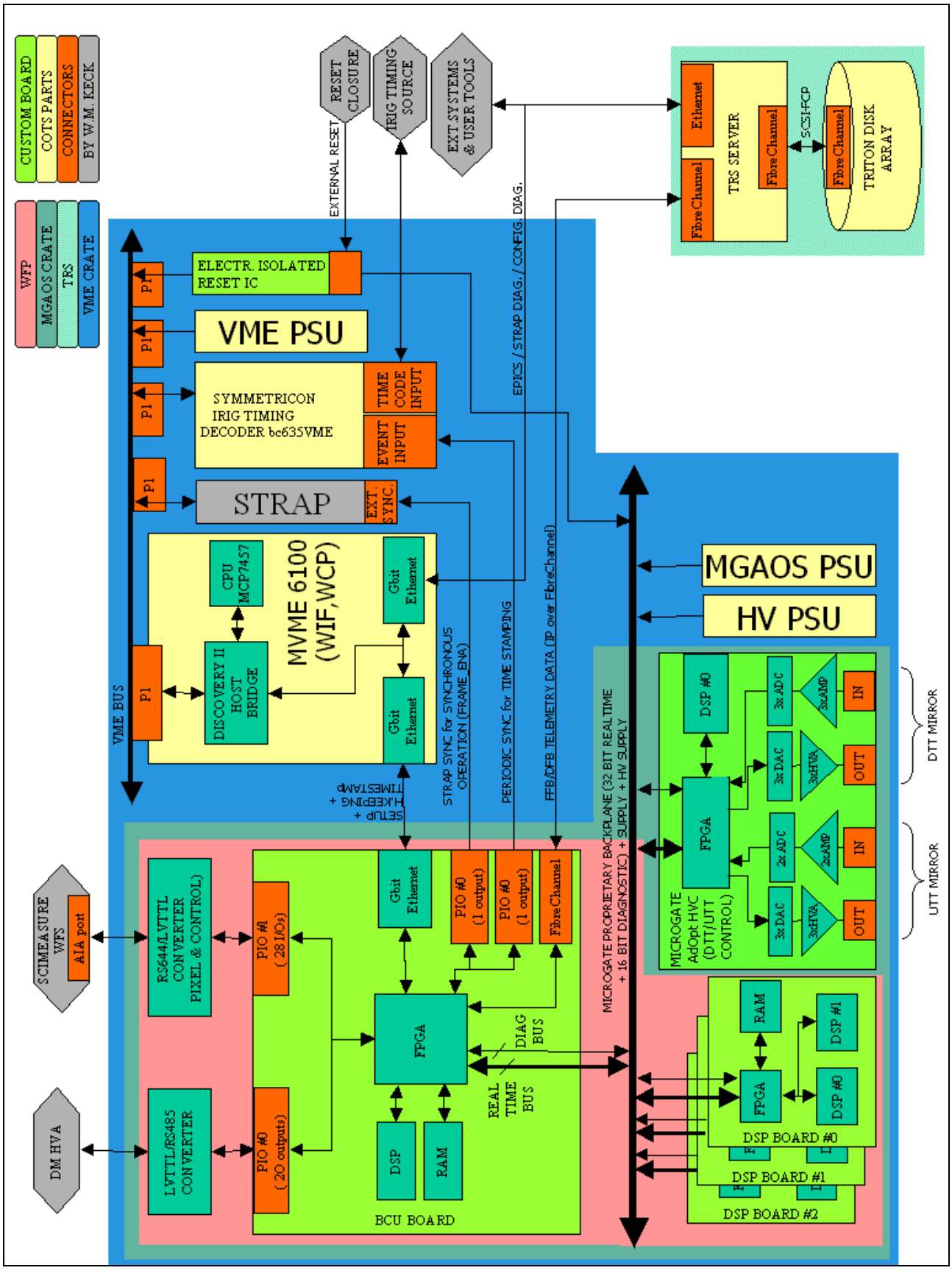


Figure 1 – NGWFC RTC global architecture

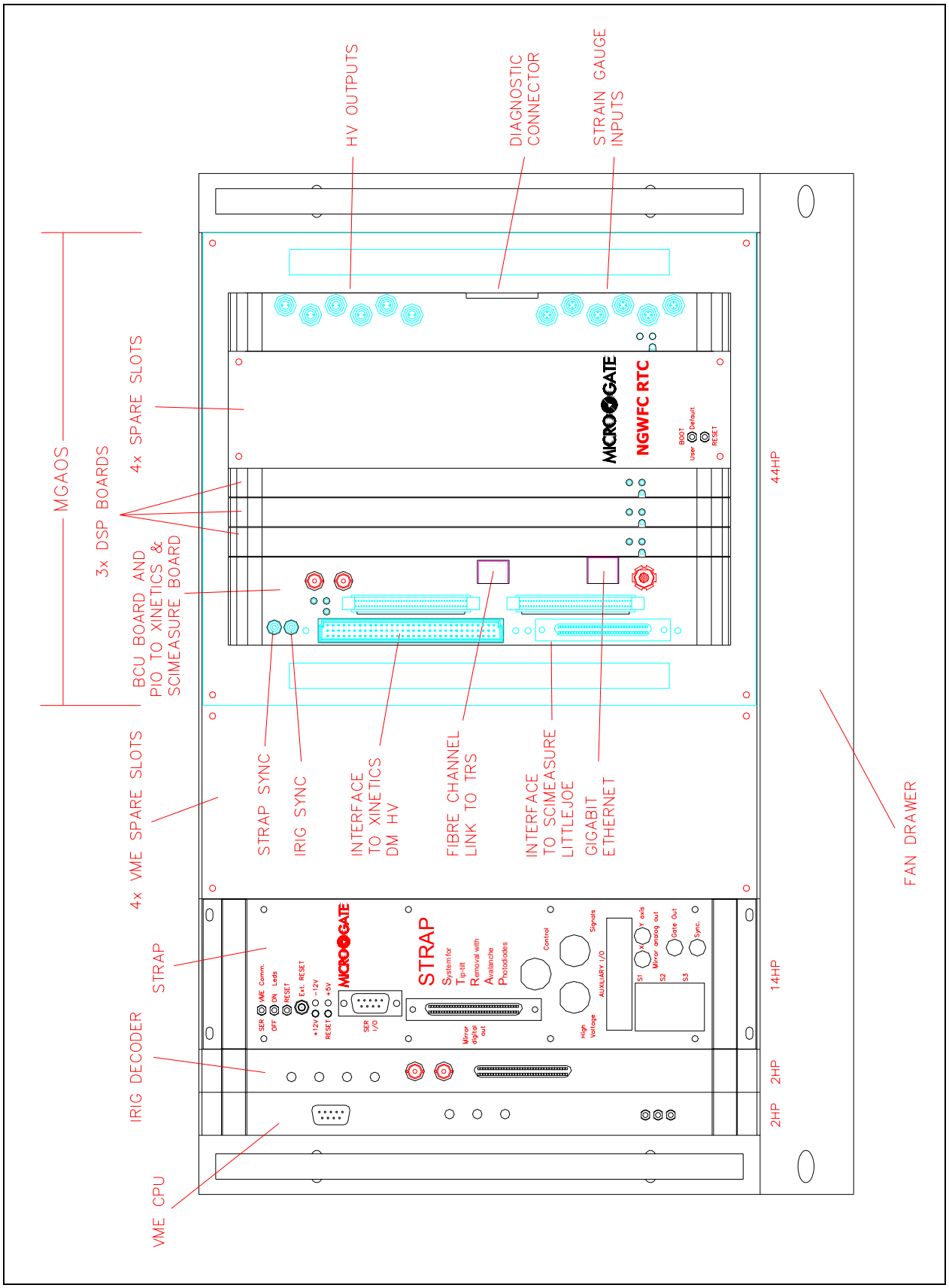


Figure 2 – NGWFC RTC: VME and MGAOS crate mechanical layout

6 HARDWARE DESIGN

In this section we describe the HW design of the NGWFC RTC. To this aim, the main system components are analyzed, namely WIF/WFP hardware, TRS and storage disk array. The subsystem main components, in particular the boards comprehended in the MGAOS system, are described in detail. We analyze also the design/simulation/prototyping results of DTT/UTT HVC board. The design process of this dedicated hardware component has been completed, including experimental tests on a single-channel breadboard. The construction will start immediately after DDR.

Finally, we describe the internal and external system interfaces.

6.1 WIF/WFP Hardware

The WFP is the real-time part of the NGWFC RTC. It acquires the pixels data from the WFS, computes the centroids, generates the residual wavefront and generates the commands for both deformable mirror and DTT/UTT tip-tilt mirrors.

The WIF is the interface between the real-time system and WCP.

Both WIF and WFP hardware are hosted on a single crate, where we can distinguish the following main components:

- Microgate crate (MGAOS), comprehending also the interfaces to WFS, DM drive and DTT/UTT mirrors. All strictly real-time WFP functions are implemented by the MGAOS
- Host VME CPU board
- IRIG timing board
- STRAP
- Power supply system

6.1.1 Microgate Adaptive Optics crate (MGAOS)

The MGAOS implements all real time functions of the WFP. The system is based on the Microgate proprietary ‘AdOpt’ hardware. The proposed configuration comprehends:

- 1 AdOpt BCU board (§6.1.1.1), implementing several functions:
  - input real-time interface to the SciMeasure controller
  - pixel background and flat field compensation
  - centroid computation
  - output real-time interface to the Xinetics controller
  - bus arbiter and master of the data exchange between different processors on the backplane (in particular, transferring centroids to the DSP boards and retrieving DM commands)
  - interface to the VME crate for real-time data transfer of STRAP data and system setup, through a dedicated Gbit Ethernet port
  - telemetry data interface to the TRS through FibreChannel port
- 3 AdOpt DSP boards (§6.1.1.2), each comprehending 2 TigerSharc DSPs (ADSP TS101) with 300 MHz internal clock rate, capable of 0.6 Giga-floating point multiply-and-accumulate per second (sustained performance, each DSP). The DSP boards implement the most extensive computational tasks, namely the residual wavefront computation and the control law servo computation.

- 1 AdOpt HVC board (§6.1.1.3), capable of controlling digitally in closed loop up to six axes based on piezoelectric transducers with strain gauge position feedback. The HVC board is used to drive directly both the DTT and UTT mirrors.
- Passive proprietary backplane (§6.1.1.3.5), based on B-LVDS technology. This is the backbone of each half-crate. Power supply, diagnostic signals, real time communication and diagnostic communication are distributed by the backplane. The backplane can host up to 12 boards, while the current configuration uses 7 slots.

The design philosophy of the AdOpt components is aimed to realize a very efficient real-time parallel computer, with particular focus on inter-processor communication.

In the following sections we describe in more detail the architecture of the main MGAOS components.

6.1.1.1 AdOpt BCU board

The BCU (Basic Computational Unit) board is a general purpose board, capable of different tasks within an adaptive optics system. It is mainly dedicated to route and manage the communication within a single Microgate AdOpt crate or among several crates or subsystems.

In the NGWFC RTC application, the BCU acts as sequencer of the different computational tasks, it manages the real-time data exchange among the MGAOS DSPs and handles the telemetry buffering and interface.

The on-board DSP handles the pixel background, flat field compensation and centroids computation tasks.

In the following section we give an overview of the main board components. A block diagram of the BCU is presented in Figure 3.

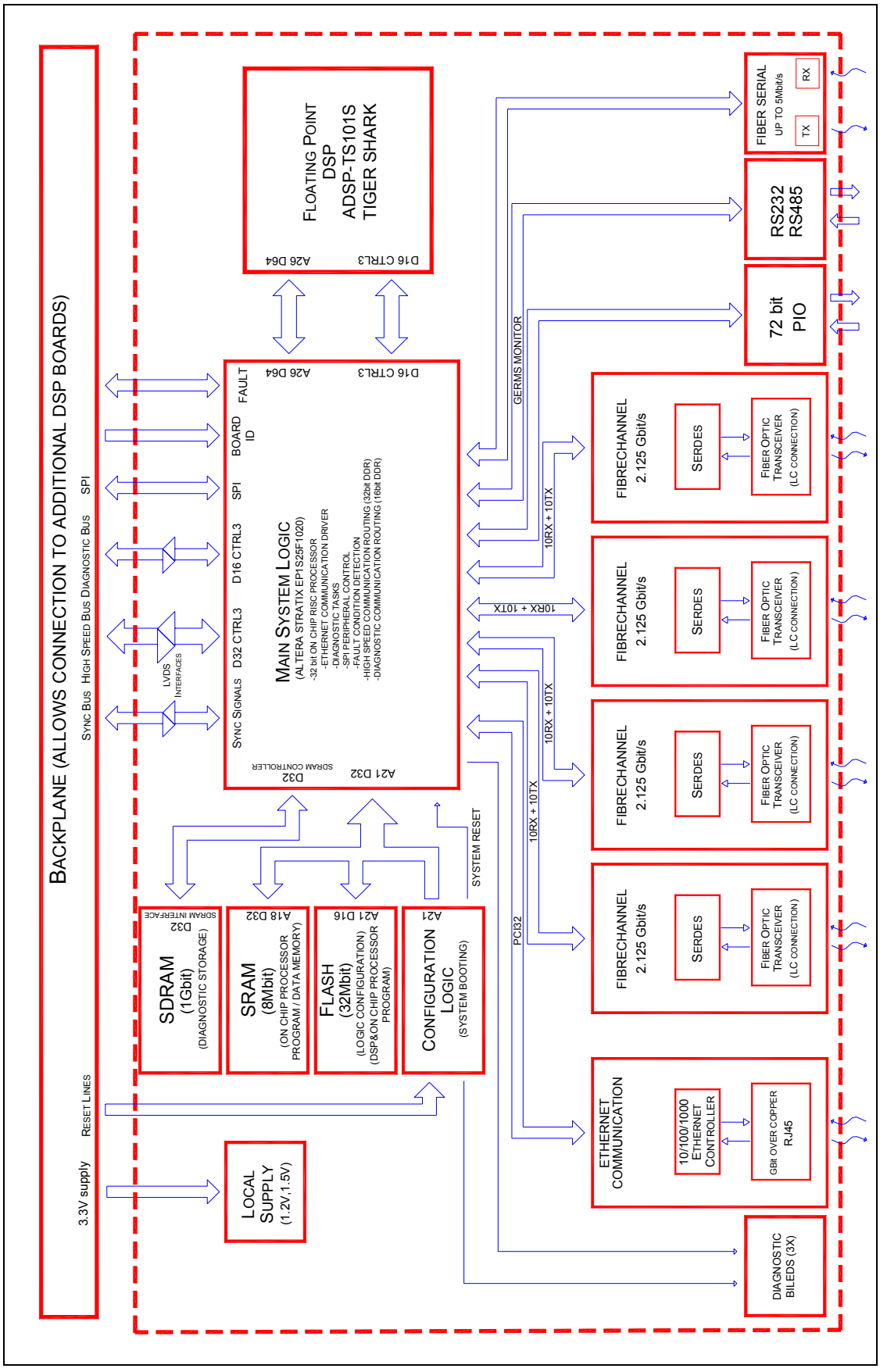


Figure 3 – BCU board block scheme

6.1.1.1.1 Main system logic

The Main System Logic is realized with an Altera Startix EP1S25F1020 programmable logic. It performs the following main tasks:

- interface between all on board devices;
- interface with the backplane buses;
- interface to the PIO (see 6.1.1.1.11). This ports provides a versatile interface to different devices. In our application it is used to read frames from the WFS and to send commands to the DM HVA
- on chip 'soft' processor (Altera Nios II), which performs:
- driver of the Ethernet diagnostic link
- acquisition of all local diagnostic devices (board temperatures)
- transfers telemetry data from DSPs to SDRAM and then to TRS

6.1.1.1.2 Main real time computational unit

The computational unit is based on one ADSP-TS101S DSPs.

The DSPs communicate with the main logic by two shared buses and some direct lines:

- the high speed bus has 64 data bits at ~60MHz of frequency with a typical throughput of 4Gbit per second;
- the diagnostic bus uses two 8 data bits parallel serial lines (called link ports) in DDR mode at ~60MHz of frequency, with a typical throughput of 2Gbit per second;
- additional ports are used by the system logic to synchronize the DSP operation with the external tasks

Within the MGAOS, the BCU DSP performs pixel background, flat field compensation and centroid computation.

6.1.1.1.3 Reconfiguration logic

The reconfiguration logic is realized with an Altera EPM3128. It is a non volatile programmable logic which has the task of downloading the configuration of the main logic and the main logic on-chip processor code from the non volatile Flash.

A safety mechanism is provided in order to guarantee that the system can be always recovered from any wrong configuration. The configuration process runs according to the following steps:

- after reset, the configuration logic starts reading the 'user' configuration from the Flash memory
- if the user configuration is not recognized by the logic as a valid configuration (validation mechanisms are factory embedded in the logic configuration loading circuitry), the configuration logic automatically loads the default configuration. This configuration is stored into a protected area of the Flash memory. Thus, it is not possible to damage accidentally the default configuration through the Ethernet connection.
- If the user configuration is recognized by the system logic as a valid configuration, but it does not work properly (as it might happen frequently during system debug), it is possible to force the default configuration by means of a dedicated hardware signal, which is available to the user (bus_boot_select)

The configuration logic also manages all the reset signals to all the devices that could be reset: Flash, main logic, DSP and Ethernet chip.

6.1.1.1.4 Non volatile storage memory (FLASH)

The non volatile memory is necessary at system booting to configure the system. The memory used is a typical flash memory with a programmable write locked area with a total size of 32Mbit.

The main logic configuration and the on-chip program code are duplicated into two different areas: the **default** and **user** area.

The default configuration and program are placed in the locked area to avoid accidental overwriting. This configuration permits to restart the system if the user configuration or program code have been corrupted or they don't work properly.

The selection of the booting mode is done either automatically, if the user mode configuration fails, or by the external hardware signal bus_boot_select.

The DSP program code area contains the DSP program. It can be automatically downloaded at system booting. In the MGAOS application, however, it is foreseen to upload the DSP code from an external database at every system startup. This task is performed by the VME CPU.

The configuration parameters is a fixed area which contains all the software parameters that define the system board configuration after start up. They are programmable using the diagnostic interface (Ethernet) and are maintained until new configuration parameters are written.

The flash memory is organized as follows:

Start address(byte)	size(byte)	locked	Description
0x000000	0x100000	yes	main logic default configuration
0x100000	0x100000	no	main logic user configuration
0x200000	0x020000	yes	on-chip default program code
0x220000	0x100000	no	on-chip user program code
0x320000	0x040000	no	DSPs program code
0x360000	0x090000	no	not used area
0x3F0000	0x00A000	no	configuration parameters
0x3FA000	0x006000	no	not used area

Table 1 – Flash memory mapping

The connection between the main logic and the flash it realized by a 16bit data bus at 60 MHz of frequency with a typical throughput of 1Gbit per second. This bus is shared with the SRAM bus because, usually, the flash device is used just at start up.

6.1.1.1.5 SRAM memory

The 8Mbit SRAM memory is dedicated only to the main logic on-chip program and data memory. The organization of the memory depends on the on-chip program code.

The connection between the main logic and the SRAM is realized by a 32bit data bus at 60 MHz of frequency with a typical throughput of 2Gbit per second. This bus is shared with the FLASH device because during ordinary operation the FLASH is never used and then the bus is free to be used with the SRAM.

6.1.1.1.6 SDRAM memory

A bulk memory is provided on the BCU board for diagnostic data storage. The bulk memory available is 1Gbit. The bulk memory is directly connected to the main system logic. It can be accessed directly by the embedded diagnostic processor and, via system logic, by the DSP. The SDRAM is connected to system FPGA by means of a 32 bit bus at 60 MHz with to Gbit/s bandwidth.

6.1.1.1.7 High speed backplane bus

The high speed backplane bus has been designed to allow the connection between the BCU board and all the control boards. The backplane bus is based on a strict master - slave architecture where the BCU board is the master and starts the transaction sending to the all control boards a request. Depending on the request all DSP and HVC boards react and can either receive the data sent by the BCU board (data write to the DSP board), or they can take the control of the bus if data have to be sent back to the BCU board.

The high speed backplane bus controller is performed by the main logic which redirects all the data from or to this bus to the high speed communication link interface.

The bus is physically based on BLVDS devices. This architecture guaranties signal integrity with high density and high speed busses.

The high speed backplane bus is a 32bit data bus at 60MHz in DDR mode with a typical throughput of 4Gbit per second.

6.1.1.1.8 Diagnostic backplane bus

The diagnostic backplane bus has been designed with a very similar architecture of the high speed backplane bus. It is controlled by the main logic which redirects all the data from or to this bus to the diagnostic communication link.

The diagnostic backplane bus is a 16bit data bus at 60MHz in DDR mode with a typical throughput of 2Gbit per second with a double speed with respect to the Ethernet diagnostic communication link. This choice has been taken to have a large margin between the internal bandwidth and the external one.

6.1.1.1.9 High speed communication links

The high speed communication allows to exchange data between different BCU boards or between BCU and external devices. In the NGWFC RTC, one high speed communication channel is used to transfer telemetry data from the BCU board to the TRS.

Every high speed communication channel is based on a full-duplex fiber optic link operating at 2.125 Gbit/s, according to the 2 Gbit/s FibreChannel physical layer standard.

Up to four high speed communication modules can be installed on the board. The number and the logic function of each channel can be configured according to the particular need.

The modules are based on the Agilent HFBR 5923 optical transceiver module (or equivalent). The module is compatible with industry standard LC-duplex fiber optic connectors, and is compatible with both 62.5-125µm and 50-125µm Multi Mode Fibers. The maximum specified link length is 150m with 62.5-125µm fiber and 300m with 50-125µm MMF.

6.1.1.1.10 Diagnostic communication link

The diagnostic communication is intended to be used for different purposes:

- transferring the diagnostic information (like acquired circular buffers and statistics);
- sending commands and acquiring system status;
- perform system maintenance.

The diagnostic communication link is based on a standard Ethernet interface, operating at 10/100/1000 Mbit/s. The physical interface is based on the standard copper link, with standard RJ 45 connector.

6.1.1.1.11 Expansion programmable input/output ports

The BCU/Communication boards comprehends a flexible expansion port (referenced as PIO – Programmable Input Output port) that can be used for interfacing different devices. The expansion ports comprehends 72 reconfigurable digital I/O lines, that can be configured according to different hardware interfacing standards (LVTTL, SSTL-2, SSTL-3, LVDS, …). The I/O lines are directly connected to the system logic (FPGA), and are available on the BCU front panel, on two 80 pins high density ribbon connectors (type 3M 810 series).

Within the MGAOS, the both PIO ports are used:

- PIO #0 implements the interface to the DM HVA. A simple interface board to convert the signal levels from LVTTL (PIO) to RS485 (DM HVA).
- PIO #1 implements the interface to the WFS. A simple interface board to convert the signal levels from RS644 (SciMeasure Little Joe AIA interface) to LVTTL (PIO).
- Two additional signals on PIO #0 are used to generate the synchronization signals for STRAP and IRIG board.

In both cases, the interfaces greatly benefit from the direct FPGA connection. In fact the FPGA handles directly the data transfers, thus offloading completely the DSP. The transfer between logic and DSP occurs by means of a true DMA process.

6.1.1.1.12 Serial links

Two standard serial communication links are available on the BCU board:

- one link is used to control and check the main logic functionalities and on-chip program flow during the system software development. This link is not used during normal operation of the system.
- the second link is available on front panel and can be used as general purpose serial interface. The serial port can be externally configured (by a configuration pin) according to RS232 or RS485 standard by means of a configuration pin. The maximum communication speed is up to 1 Mbit/s. There is no use foreseen for this port in the MGAOS.

6.1.1.1.13 ‘Slow’ fiber input/output

An additional fiber link is available on the BCU board. This port can be used for different purposes:

- asynchronous serial interface with speed up to 5 Mbit/s
- bi-directional transmission of logic signals, e.g. for synchronization purposes.

The fiber optic interface has ST standard connectors. The transmitter/receiver sections are based on Agilent HFBR 1412/2412 devices. One important feature is that the optical interface is ‘static’, i.e. there is no lower speed limit for the information or signals to be transmitted.

In the present design, there is no use foreseen for the ‘slow’ fiber interface.

6.1.1.1.14 Power supply

The BCU board requires a single 3.3V @ 3A (typical) supply. Internally, additional supply voltages are required for board operation, in particular 1.2V and 1.5V at relatively high currents for the DSP and FPGA cores respectively. The high accuracy, low impedance requirements on these power rails can be more easily satisfied if a distributed supply concept is adopted, where these low voltages are generated on-board by means of high efficiency, fast switching DC-DC converters.

The internal supply distribution scheme is resumed in Table 2.

Supply of:	Voltage	Current	Generated by:
DSP core	1.2V	1.2A	On-board switching DC-DC converter, derived from 3.3V
FPGA core	1.5V	0.8A	On-board switching DC-DC converter, derived from 3.3V
Logic, BLVDS	3.3V	0.85A	3.3V (VCC)
Gigabit Ethernet module	3.3V	1.2A	3.3V (VCC)
FibreChannel modules	3.3V	0.8A	3.3V (VCC)

Table 2 – BCU/Communication board supply distribution

6.1.1.1.15 Direct backplane bus signals

From the backplane bus there are some directs signals controlling some hardware configurations or time critical events. The following table summarize these signals:

Name	Direction	Description
bus_power_fault	in/out	typically configured as an input to verify if an external power fault condition happens. If a fault condition happens inside the board the signal is driven as an output by the main logic to notify the fault condition to the rest of the system
bus_sys_rst_n	input	performs a global reset of the board
bus_dsp_rst_n	output	generates a global reset to all DSP and HVC boards (can be SW generated internally by SW)
bus_fpga_clr_n	input	performs a soft reset of the board logic
bus_driver_enable	Input/output	Used in HVC board to force unconditional disabling of high voltage drives (emergency situation). The signal can be monitored by the BCU and forced to disable drives under software control.
bus_boot_select	input	selects the default or user main logic configuration and on-chip program code at the board booting
bus_slot_id[0 .. 3]	input	three static bits and one “three state” bit to identify the slot position from 0 to 31 where the control board is inserted

Table 3 – Direct backplane bus signals

6.1.1.1.16 High speed communication description

The high speed communication is used to transfer the real time data between different parts of the MGAOS. The typical communication paths are shown on Figure 4.

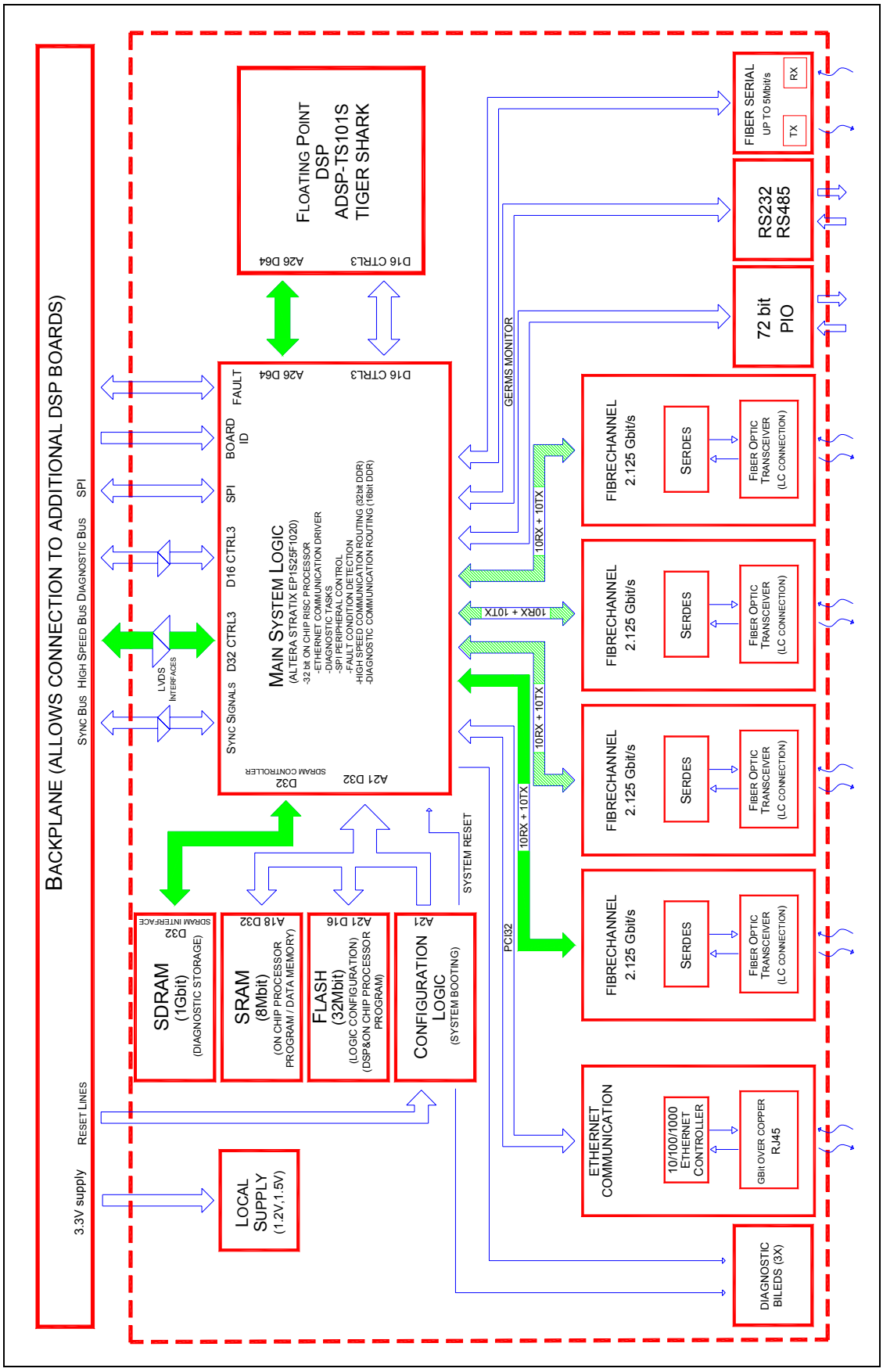


Figure 4 – High speed communication paths

Data can be sent or to received from the high speed communication inputs or exchanged among DSPs. In the system logic, data packets are processed and, according to the message destination, are dispatched to the final receiver. This might be the local DSP, other DSPs within the same crate, another receiver on a different BCU board (not in the current application) or an external device (in our application, the TRS HBA. In this implementation data are read from BCU SDRAM).

The communication from/to the high speed interface modules goes through independent busses (10 bit wide, separated reception and transmission).

When data are sent to or received from the local DSP, the synchronous communication mode on the 64 bit parallel bus of the DSP is used. Data are transferred through a DMA channel on the DSP, therefore the data flow does not interfere with the DSP processing activity. Latencies can occur only if another DMA process attempts to access simultaneously to the same memory location in the DSP internal memory, which is an extremely rare condition. It is important to notice that the DSP parallel bus is used only for real time communication (this is true also for the DSP and HVC control boards).

If the data are destined to other DSPs on the same crate, the logic transfers the data through the 32bit bus on the backplane. Similarly as above, also this bus is dedicated only to real time data transfer.

The communication speed on both the DSP parallel bus and on the 32bit backplane bus is of 4Gbit/s.

6.1.1.1.17 Diagnostic communication description

Diagnostic communication is destined to transfer commands and large amounts of diagnostic data with less stringent speed requirements. Diagnostic communication passes through the Ethernet port, which is connected by a dedicated PCI bus (32 bit) to the on-chip diagnostic processor in the system logic.

Diagnostic data might be transferred from/to the local DSP. To this aim, a dedicated interface, based on the DSP high speed synchronous communication ports (LinkPort) has been foreseen. The transfer rate is up to 2Gbit per second. This link does not interfere with the DSP bus activity, that supports the high speed communication, and can be also handled by DMA processes.

When diagnostic data need to be transferred to other DSPs on the same crate, a dedicated 16bit parallel bus is available on the backplane. This bus is similar to the high speed communication one (but 16bit wide), and has a throughput of 2Gbit per second.

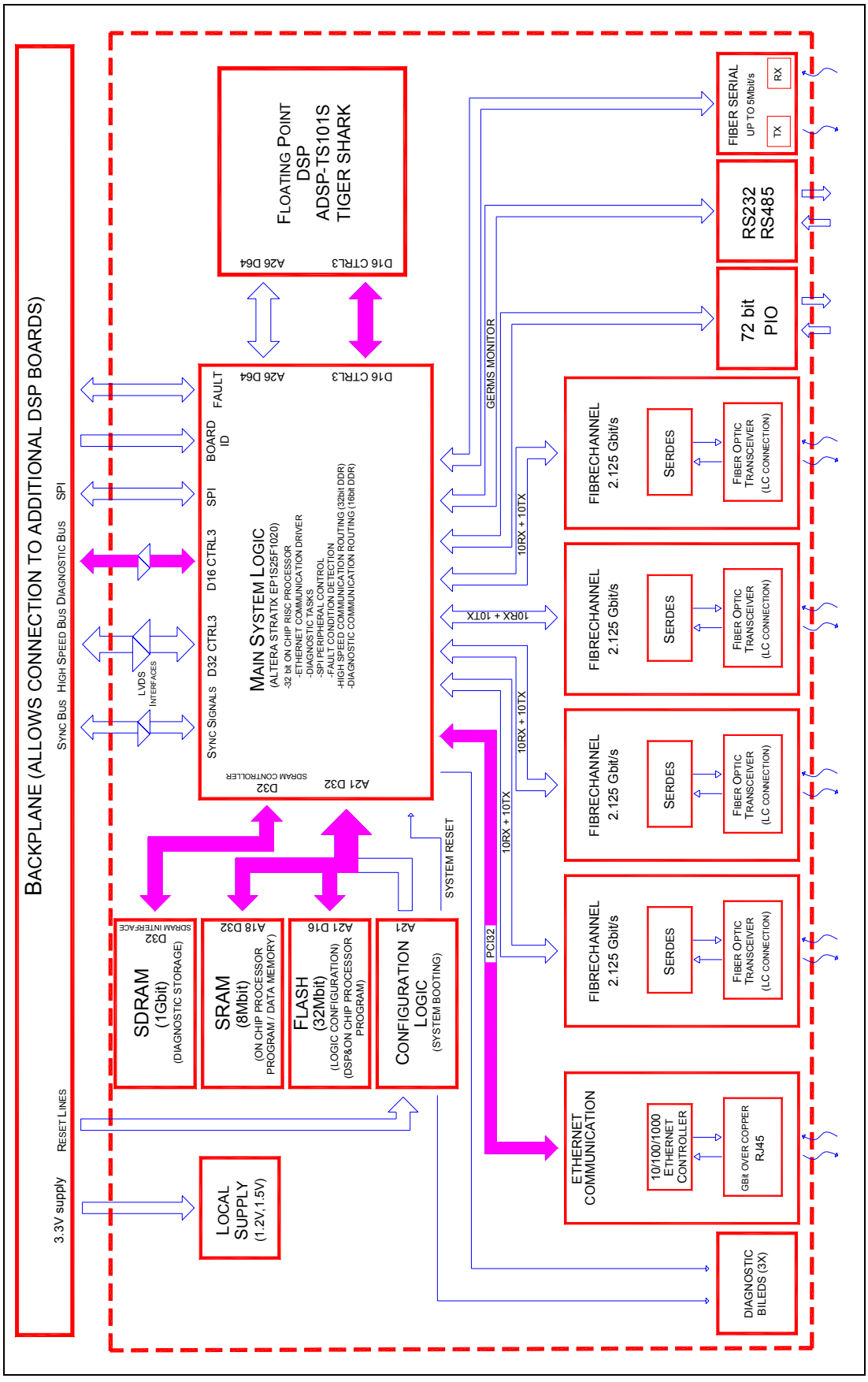


Figure 5 – Diagnostic communication paths

6.1.1.1.18 Boards layout

The BCU board PCB size is 196x91 mm. The limited available space forced to select components with very small SMD packages. Most of the components are mounted on the upper side of the board. The use of the bottom side is limited to decoupling capacitors and some analog components.

The components placing has been structured in functional zones. With reference to Figure 6, we can distinguish the following areas:

- Logic and computation: DSPs, FPGA, memories, configuration logic
- link communication front-end;
- Bus interface: B-LVDS drivers;
- Power supply: on-board switching and linear regulators;

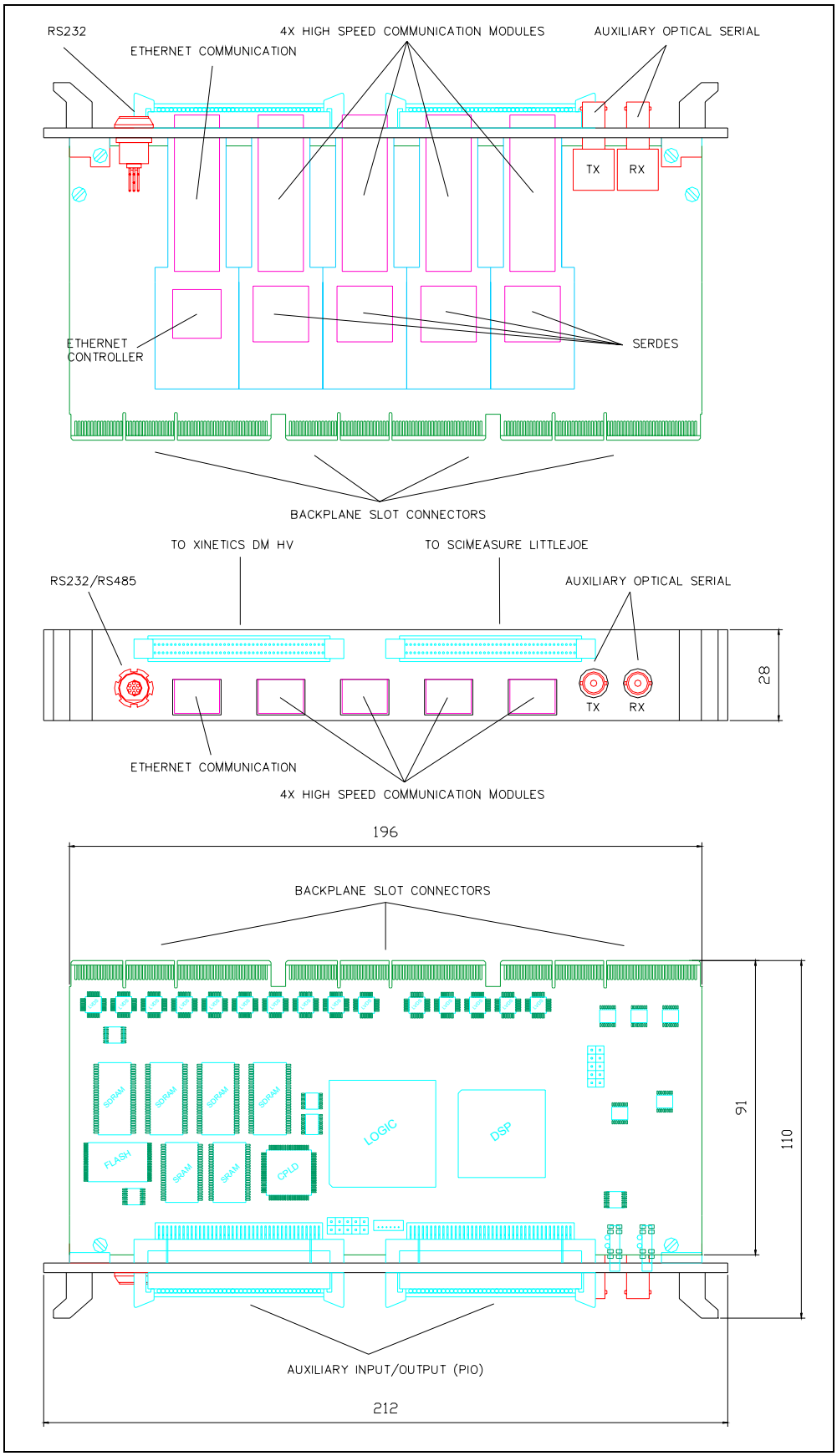


Figure 6 – BCU board components placing, with evidence of the different functional zones.

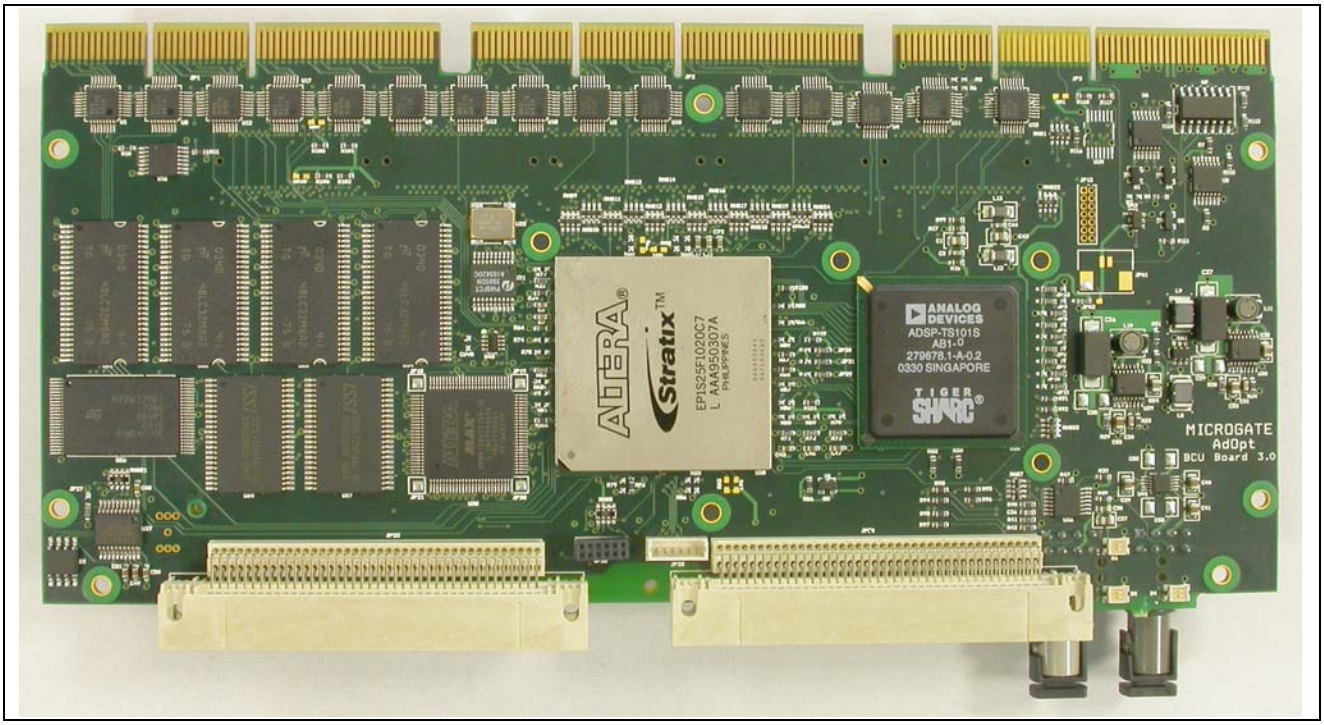


Figure 7 – BCU board.

6.1.1.2 AdOpt DSP BOARD

The *DSP board* is a very important component within the MGAOS. It performs the most heavy real time computations, namely residual wavefront computation and the control law servo computation. The total computational load is distributed in a parallel way among the 3 DSP boards foreseen within the system.

In other adaptive optics applications, the DSP board is directly interfaced to analog actuators and sensors. The analog section of the DSP board is not mounted on the boards dedicated to real-time computation only.

As an exception, the analog part is used for the DSP board that is part of the HVC board, as described in §6.1.1.3.

6.1.1.2.1 DSP board block scheme

The block scheme of the DSP boards is presented in Figure 8.

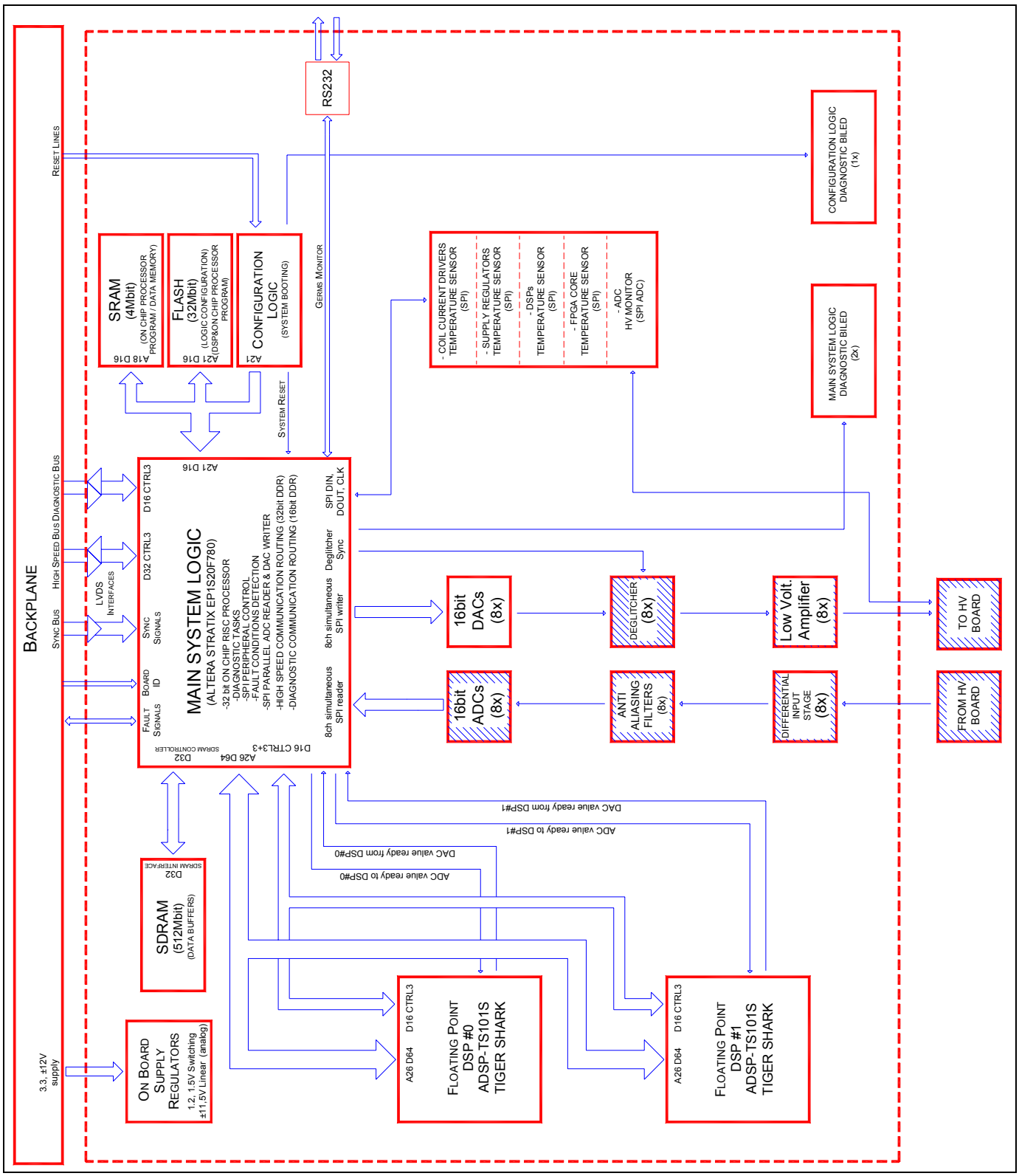


Figure 8 – DSP boards block scheme. The shaded parts refer only to the DSP board mounted on the HVC board.

As shown in Figure 8 the DSP board architecture is designed around the Main System Logic. All data transfers within the board pass through the main system logic which creates the correct interface between the devices.

6.1.1.2.2 Main System Logic

The Main System Logic is realized with an Altera Startix EP1S20F780 programmable logic. It performs the following main tasks:

- the interface between all the on board devices;
- the interface with the backplane buses;
- the on chip processor, which performs the acquisition of all local diagnostic devices, the first level fault analysis and, if necessary, takes the actions correlated;

6.1.1.2.3 Computational devices

The computational part is realized with two ADSP-TS101S DSPs, where all real-time processing is performed.

The DSPs communicate with the main logic by two shared buses and some direct lines:

- the high speed bus has 64 data bits at 60MHz of frequency with a typical throughput of 4Gbit per second; The bus is shared among the two DSPs and the main system logic.
- the diagnostic bus uses two 8 data bits parallel serial lines (called link ports) in DDR mode at 60MHz of frequency, with a typical throughput of 2Gbit per second;
- dedicated hardware flags, directly connected between DSPs and main logic, to provide a proper synchronization and detection of completion of the computational steps

6.1.1.2.4 Configuration Logic

The configuration logic has identical function as in the BCU board. Refer to §6.1.1.1.3.

6.1.1.2.5 Non volatile storage memory (FLASH)

The configuration logic has identical function as in the BCU board. Refer to §6.1.1.1.4.

In the HVC board, the FLASH memory contains also the calibration parameters (offset and gain) for each analog input/output channel.

6.1.1.2.6 SRAM memory

The 4Mbit SRAM memory is dedicated only to the main logic on-chip program and data memory. The organization of the memory depends to the on-chip program code.

The connection between the main logic and the SRAM is realized by a 16bit data bus at 60 MHz of frequency with a typical throughput of 1Gbit per second. This bus is shared with the FLASH device because during ordinary operation the FLASH is never used and then the bus is free to be used with the SRAM.

6.1.1.2.7 SDRAM memory

The 512Mbit SDRAM is used to storage real time data for diagnostic purposes. The main logic manages the bi-directional transfer of diagnostic data between SDRAM and DSPs' memory. Transfer from/to DSPs' memory is performed using a true DMA channel, therefore there is no software overhead by the DSP.

The data flow from the DSPs to the SDRAM is routed through the diagnostic channel, i.e. 16 bit 60MHz DDR LinkPort on DSP and 32 bit, 60MHz synchronous bus on SDRAM. The typical throughput is 2Gbit per second.

6.1.1.2.8 High speed backplane bus

The same considerations as in §6.1.1.1.7 apply to the high speed bus interface with the backplane. The DSP board is slave in the high speed communication. Internally, real time data are routed directly to/from both DSPs using the 64 bit DSP bus. Data transfers are handled by logic only and DSP memory is accessed in DMA, without requiring any software overhead. The high speed backplane bus is a 32bit data bus at 60MHz in DDR mode with a typical throughput of 4Gbit per second.

6.1.1.2.9 Diagnostic backplane bus

The same considerations as in §6.1.1.1.8 apply to the diagnostic bus interface with the backplane.

6.1.1.2.10 The diagnostic serial monitor

A standard RS232 serial link allows to control and check the main logic functionalities and on-chip program flow during software development. This link is not available to the user and it is not foreseen to use it for ordinary operation of the board.

6.1.1.2.11 Direct backplane bus signals

From the backplane bus there are some directs signals which controls some hardware configurations or time critical events. The following table summarize these signals:

Name	Direction	Description
bus_power_fault	in/out	typically configured as an input to verify if an external power fault condition happens. If a fault condition happens inside the board the signal is driven as an output by the main logic to notify the fault condition to the rest of the system
bus_sys_rst_n	input	performs a global reset of the board
bus_dsp_rst_n	input	performs a global reset of the board, similar to the bus_sys_rst_n from the control board point of view
bus_fpga_clr_n	input	performs a soft reset of the board
bus_boot_select	input	selects the default or user main logic configuration and on-chip program code at the board booting
bus_driver_enable	input	Used in HVC board to force unconditional disabling of high voltage drives (emergency situation)
bus_slot_id[0 .. 3]	input	Three static bits and one “three state” bit to identify the slot position from 0 to 31 where the control board is inserted

Table 4 – Direct backplane bus signals

6.1.1.2.12 Power supply

The DSP board requires a single 3.3V @ 1.5A typ. Other voltages are required for board operation, in particular 1.2V and 1.5V at relatively high currents for the DSP and FPGA cores respectively. The high accuracy, low impedance requirements on these power rails can be more easily satisfied if a distributed supply concept is adopted, where these low voltages are generated on-board by means of high efficiency, fast switching DC-DC converters.

The internal supply distribution scheme is resumed in Table 5.

Supply of:	Voltage	Current	Generated by:
DSP core	1.2V	1.2*2=2.4A	On-board switching DC-DC converter, derived from 3.3V
FPGA core	1.5V	0.6A	On-board switching DC-DC converter, derived from 3.3V
Logic, BLVDS	3.3V	0.65A	3.3V (VCC)

Table 5 – DSP board supply distribution

6.1.1.2.13 Mechanical configuration

The DSP board PCB size is 196x91 mm. The limited available space forced to select components with very small SMD packages. Most of the components are mounted on the upper side of the board. The use of the bottom side is limited to decoupling capacitors and some analog components.

The components placing has been structured in functional zones. With reference to Figure 9, we can distinguish the following areas:

- Logic and computation: DSPs, FPGA, memories, configuration logic
- Analog input: capacitive sensors input amplifier and ADCs (HVC board only)
- Analog output: DACs and low voltage amplifiers (HVC board only)
- Bus interface: B-LVDS drivers
- Power supply: on-board switching and linear regulators

The adopted scheme allowed to obtain a very efficient routing, in order to limit cross-talk problems.

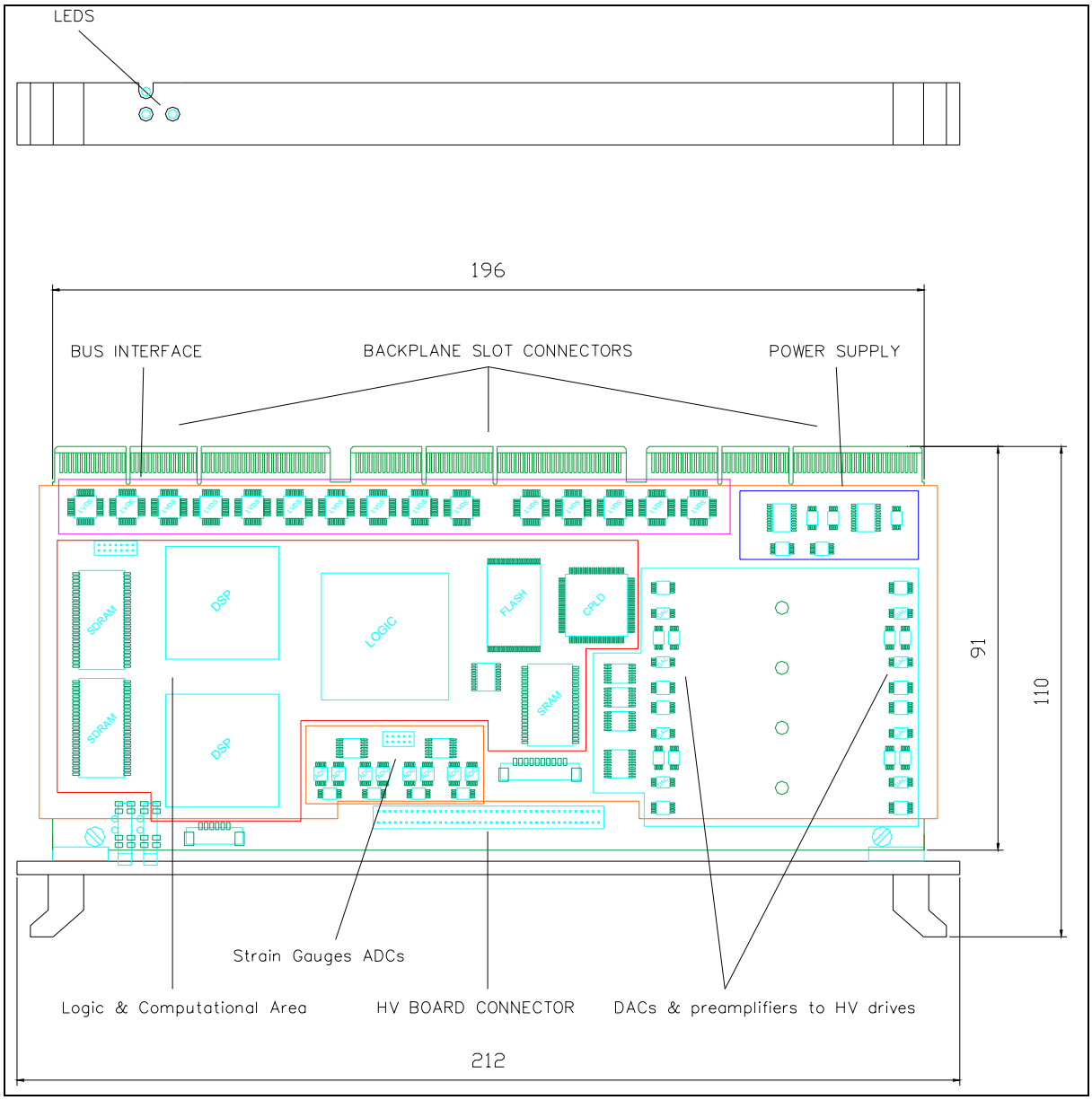


Figure 9 – DSP board components placing, with evidence of the different functional zones.

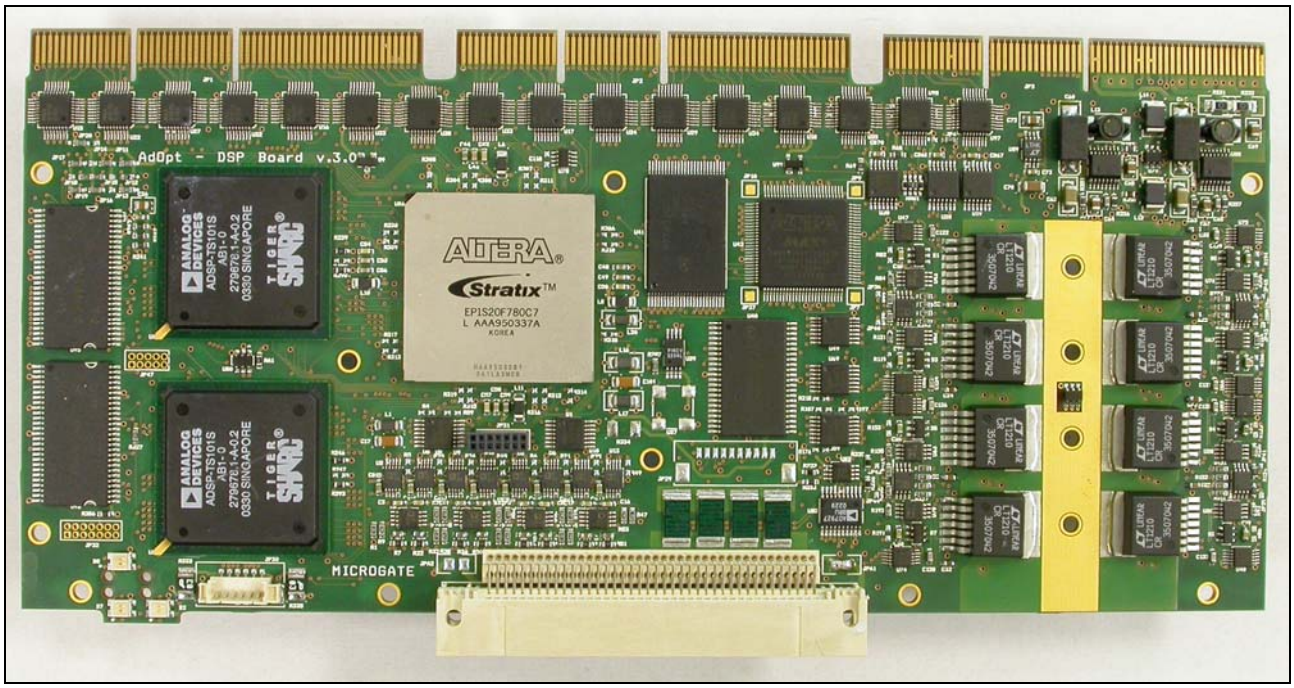


Figure 10 – DSP board V3.0. The board in this picture is completely mounted, including the analog parts used only on the HVC board.

6.1.1.3 AdOpt HVC board

The purpose of the HVC board is to drive and control in open and closed loop the Tip-Tilt mirror units (DTT and UTT).

Within the HVC board, we can distinguish two main subsystems:

- ***digital and low voltage analog control board***. This part is based on the AdOpt DSP board (see §6.1.1.2), with very little modifications, mainly change of some components value. This board has been specifically developed for controlling voice coil motor force actuators with a capacitive sensor position feedback. This design is very well consolidated and similar boards have been already manufactured in significant quantities.
- ***High voltage analog drives and strain gauges conditioning circuitry***, placed on a second board directly interfaced (as daughter-board) to the AdOpt DSP board.

The AdOpt HVC board has 6 independent channels.

The DTT mirror requires three driving channels and three strain gage inputs, while the UTT mirror requires just two driving channels and two strain gage inputs. One third high voltage output is required to bias the 'hot' common point of the UTT mirror. In the proposed configuration, this mirror mirror channel is driven by an additional high voltage driving channel. In this way, it is possible to control under software a 'smooth' power up and power down of the mirror (see §7.1.5.4).

6.1.1.3.1 DSP board analog input section

The input section of the AdOpt DSP board comprehends an input operational amplifier configured as quasi-differential stage and providing first order, anti-aliasing low pass filter. This stage drives a 16 bit ADC (type: AD7685). This ADC has been chosen for the following parameters:

- maximum conversion rate of 250 Ksamples/s

- very low power consumption, typ. 20mW
- 3.3V SPI serial interface
- very limited size (3x3mm)
- signal to noise ratio: 89dB

The ADC output is directly interfaced to the FPGA that generates the proper sampling signals and reads the eight ADCs simultaneously. Acquired data are directly transferred to the DSP memory without software overhead. At the end of the process, the DSP is notified using a dedicated hardware line. In this way, the readout time amounts to 800ns, while the total delay from 'start of sampling' to 'data available to the DSPs' amounts to 3µs.

6.1.1.3.1.1 DSP board analog output section

The analog reference for the piezoelectric actuators high voltage drives is generated by a 16 bit DAC type Maxim MAX5442.

This DAC has the following relevant characteristics:

- Settling time = 1µs to ½ LSB
- very low power consumption, typ. 60mW
- 3.3V SPI serial interface
- very limited size (3x3mm)
- thermal stability: 0.05 ppm/°C
- signal to noise ratio: 92dB
- digital feedthrough: 0.2nV/s
- seamless interface to generate bipolar output

The final drives reference output is buffered by a high speed, low noise, precision operational amplifier (AD8512) and connected to the high voltage drive (placed on the daughter-board, see §6.1.1.3.1.2).

With respect to the digital interface, the DAC input is directly interfaced to the FPGA that loads the new values on all eight DACs simultaneously. The data are automatically taken from the DSP internal memory. In this way, the update (excluding settling time) lasts just ~800ns.

6.1.1.3.1.2 High voltage drives daughter-board

The high voltage daughter-board comprehends the following sub-circuits:

- six high voltage drives
- six differential analog inputs acting as signal conditioner for strain gauge bridges
- strain gauges supply circuit
- high voltage 'safety' circuits
- local and external diagnostic circuits

The design of the high voltage drives has been based on the specifications reported on Table 6:

	DTT	**UTT**
Mirror type	Custom mirror based on 3x PI 841.60 piezoelectric actuators	PI S330.10 tit-tilt mirror
Capacitance (each actuator)	10.8 μF	3.6 μF
Dynamic Operating Current Coefficient	15 μA hz^{-1} μm^{-1} (actuator) 2.7 μA hz^{-1} μrad^{-1} (actuator)	0.22 μA hz^{-1} μrad^{-1}
Voltage range	0-100V	0-100V
Stroke (0-100V)	90 μm (actuator) 0.738 mrad (mirror)	± 1 mrad
Self resonant frequency	1 kHz	2.4 kHz
Dynamic	Actual mirror commands time-history provided by Keck, based on STRAP diagnostic.	

Table 6 – Tip-tilt mirror specifications

The high voltage drive is based on a linear amplifier. This configuration has been preferred to a switching typology to reduce the electromagnetic noise generated by the drive. This is particularly critical in closed loop operation, with the high sensitivity strain-gauge amplifiers placed very close to the high voltage amplifiers. Moreover, as shown in Table 10, the dissipation is very reasonable even using a linear device.

The main component of the high voltage stage, namely the high voltage operational has been selected considering various aspects, in particular:

- Voltage range
- Output current
- Dynamic response
- Quiescent current
- Package

In order to get an estimate of the **voltage and current requirements**, we have simulated the theoretical voltages and currents required to drive the DTT mirror to compensate a real turbulence pattern (generated by real acquisition at Keck), acquired in 'very bad' seeing conditions. The DTT mirror has been chosen to define the design boundaries, because of the higher actuators capacitance.

The gains used in the simulation are the actual ones currently set at the telescope.

The results are presented in Figure 11 and Table 7.

As it can be noticed, the output voltages obtained from the simulation do clearly exceed the voltage range of the drive currently in use (0-100V). To overcome this limitation, the voltage range for the new design has been specified to be significantly higher than now.

The minimum requirements for the drives have been derived according to the following criteria:

- Voltage: imposed by maximum specification for the actuators ⇒ **-20/+120V** for both DTT and UTT mirrors
- Current: greater than maximum between 6σ and min-max ⇒ **> 39.8mA**

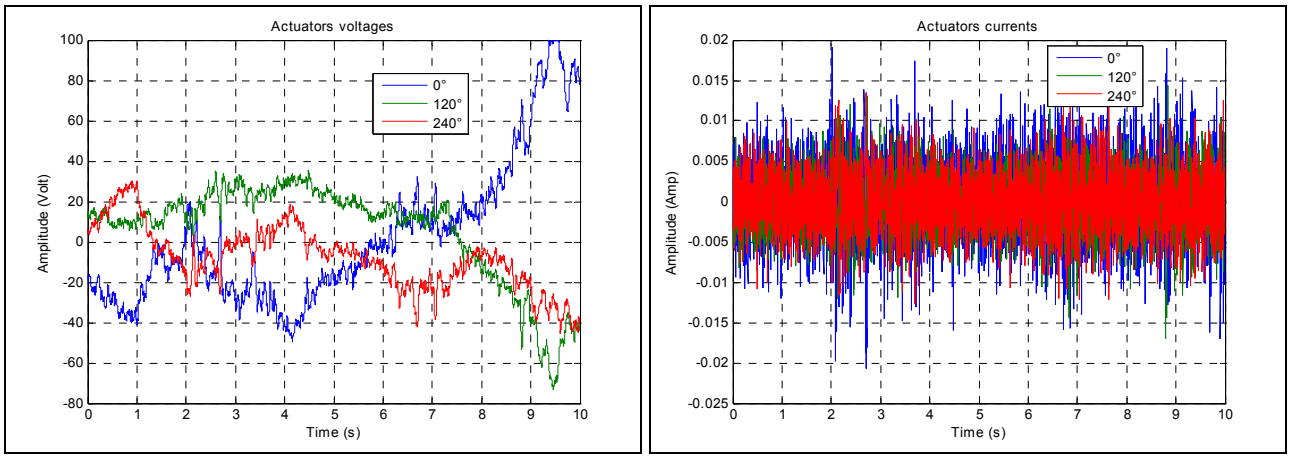


Figure 11 – DTT mirror actuators driving voltages and currents for typical 'bad seeing' compensation.

Parameter	***Act.1 (0°)***	***Act. 2 (60°)***	***Act. 3 (120°)***
Voltage min-max (V)	149.7	108.8	75.9
Voltage RMS (V)	35.6	24.3	15.7
Max slew rate (V/ms)	1.91	1.58	1.25
Current min-max (mA)	39.8	31.5	26.2
Current RMS (mA)	4.45	3.21	3.28

Table 7 – DTT mirror actuators driving voltages, slew rate and currents. RMS and min-max values for typical 'bad seeing' compensation.

With respect to the **dynamic requirements**, we can distinguish between full-power and small signal dynamics.

Full power requirements are related to possible drive saturation due to excessive current and/or slew-rate limitations. The maximum **slew rate** for the studied turbulence simulation is of **1.91 V/ms**.

The **small signal** (i.e. below any saturation) dynamic response requirement can be evaluated considering the closed loop bandwidth specification of 200Hz (see req. 248 and 282 in [AD6]).

Based on the considerations above, a suitable high voltage operational amplifier is the Apex type PA243DF.

The most relevant characteristics of these device are reported on Table 8:

Maximum supply voltage	350 V
Peak output current	120 mA
Continuous output current	60 mA
Quiescent supply current	2.2 mA
Output voltage swing	$\|\pm V_{supply}\|$ - 10
Slew rate	30 V/µs
Gain Bandwidth Product	3 MHz
Noise, 0-10kHz	50 µV RMS
Package (dual channel amplifier)	24 pin PSOP (14.5 x 16mm footprint)

Table 8 – Apex PA243 amplifier main characteristics

The static and dynamic performances are well suited for the application, with relevant margins. Low quiescent current and small package are also important factor, allowing us to design a very compact multi-channel control board.

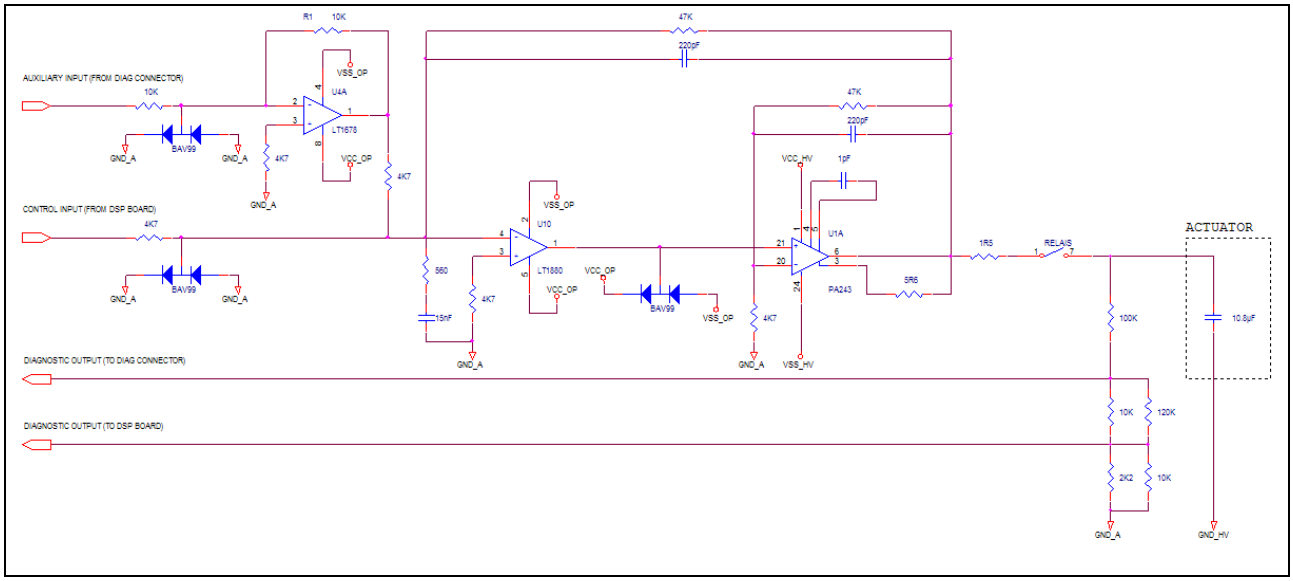


Figure 12 – High voltage drive output stage

The high voltage drive output stage is configured as composite amplifier (see Figure 12). The high voltage op-amp is driven by a precision, low noise, high speed operational amplifier (AD8512). The stabilization of the drive is obtained by means of load isolation through output resistance, dominant pole compensation on feedback network and noise gain compensation on the inverting input. All these parameters have been optimized for the DTT mirror, because its actuators are more critical to drive (10.8µF with respect to 3.6µF on of UTT mirror actuators). The gain of the composite output stage has been set to 12, in order to match both the output range of the DAC driver and the input characteristics of the existing design (a summing input allows to inject analog signals to the high voltage drive for testing purposes, see Figure 12).

The analog performances obtained by design are reported in §6.1.1.3.4.

The *strain gauge input stage* comprehends a precision, low noise differential amplifier (AD8221). The gain of this stage is 50. This first stage is connected to the input amplifier on the DSP board having a gain of 20, therefore the total gain is 1000. It comprehends also the anti-aliasing low-pass filter (dual pole at 26.5 kHz, matched with the control loop rate of 70 kHz), and common mode/differential mode filters on the strain gage inputs, to minimize the effects of noise pick-up on the transmission line.

For safety reasons, all high voltage outputs foresee a relay that allows disconnecting physically the output from the actuator. All these devices are normally open and the high voltage circuits are automatically disconnected in case of system malfunctioning (e.g. intervention of control board watchdog).

The high voltage outputs are connected to a diagnostic ADC (8 channel, 12 bit resolution) on the DSP board. These acquisition channels are not used for real-time control, they are just provided for diagnostic purposes.

The two additional channels monitor continuously the high voltage supply rails.

Additionally, analog monitor and test points are provided on all high voltage channels. These include:

- High voltage drive output voltage divided by 10 (-2 ~ 12V).
- Input signals to a summing junction of the high voltage amplifier. A gain of 10 is applied to the input signal (input range: -2 ~ 12V)
- Strain gauge signals after differential initial amplification (gain 50mV/µm).

6.1.1.3.2 Power dissipation

The HVC board requires five power supplies: 3.3V for logic, ±12V for analog circuitry and –35,+135V for high voltage drives. Other voltages are required for the operation of the DSP board, in particular 1.2V and 1.5V at relatively high currents for the DSP and FPGA cores respectively. As for the standard DSP board, these additional power rails are generated on-board. It shall be noticed that just one DSP is present on the DSP board of HVC.

The high voltage current has been estimated as follows:

- Quiescent current of HV amplifiers: 2.2mA x 6ch = 13.2mA
- Typical RMS current (bad seeing condition): 3.6mA x 6ch = 21.6mA
- Total: 34.8mA

The internal supply distribution scheme is resumed in Table 5.

Supply of:	**Voltage**		**Current**	**Generated by:**
DSP core	1.2V		1.2A	On-board switching DC-DC converter, derived from 3.3V
FPGA core	1.5V		0.6A	On-board switching DC-DC converter, derived from 3.3V
Logic, BLVDS	3.3V		0.65A	3.3V (VCC)
Analog	±11.7V		0.02A	On-board linear regulators, from 12V
High Voltage drives	-35	+135V	0.0348A	Dedicated power supply for high voltage drives.

Table 9 – HVC board supply distribution

6.1.1.3.3 Mechanical configuration

The HVC board is configured as a stack-up of two PCBs. An AdOpt DSP board, slightly modified for the scope, is connected to a second PCB where the high voltage amplifiers and the strain gauge signal conditioning amplifiers are mounted.

The mechanical layout is shown in Figure 13. The board occupies two slots of the MGAOS crate (28mm).

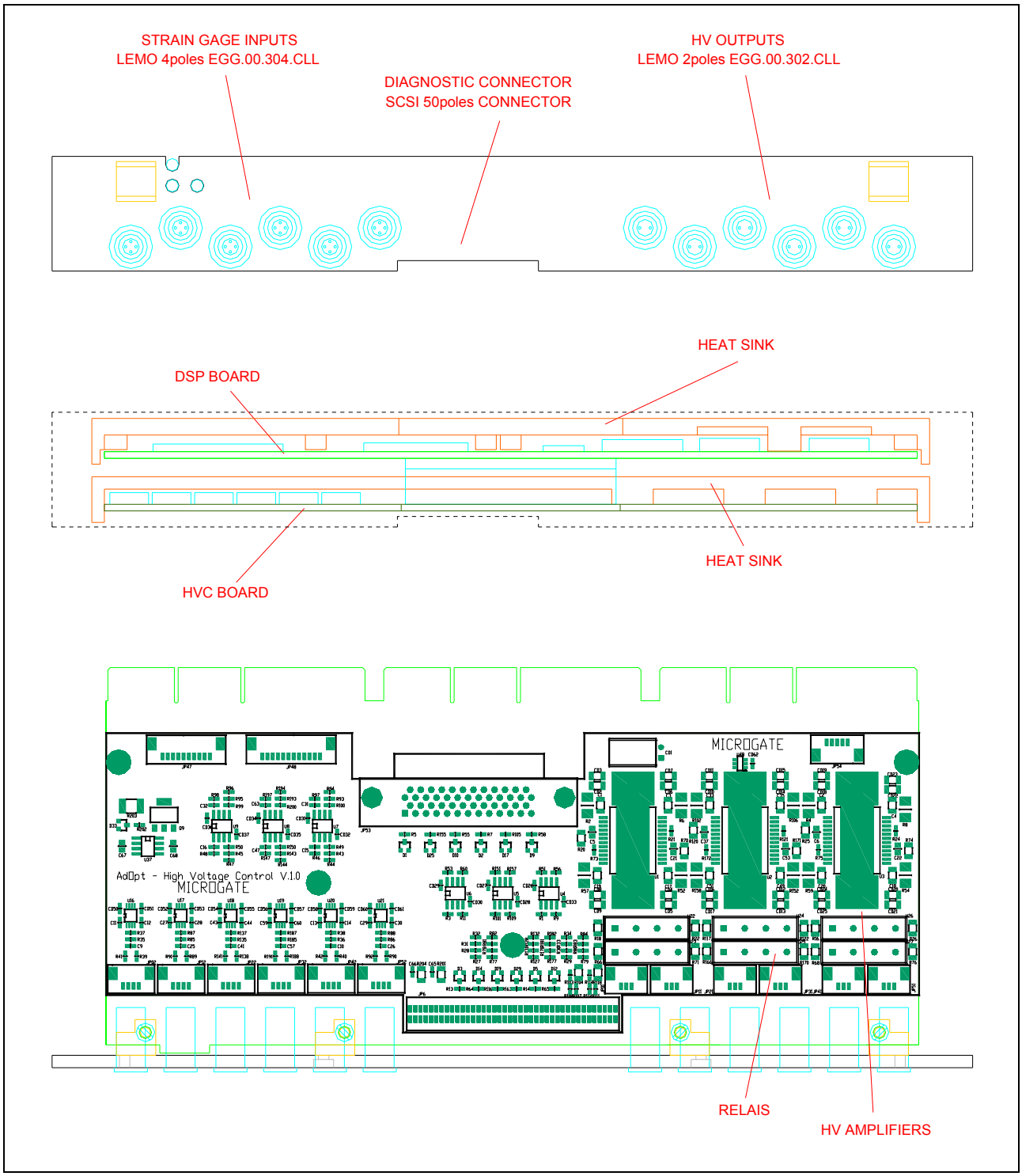


Figure 13 – HVC board mechanical layout

6.1.1.3.4 Performance estimate through numerical simulation

The performance of the HVC board has been estimated using numerical simulation.
Hereafter we report the simulation procedure and related results:

- *Simulation of single actuator high voltage drive electronics.* The electronic design has been preliminarily optimized and simulated using PSPICE analog simulator. The actuator has been represented as a capacitance. The simulation results are reported on Figure 14 and Figure 15. The obtained bandwidth is of 13 kHz.

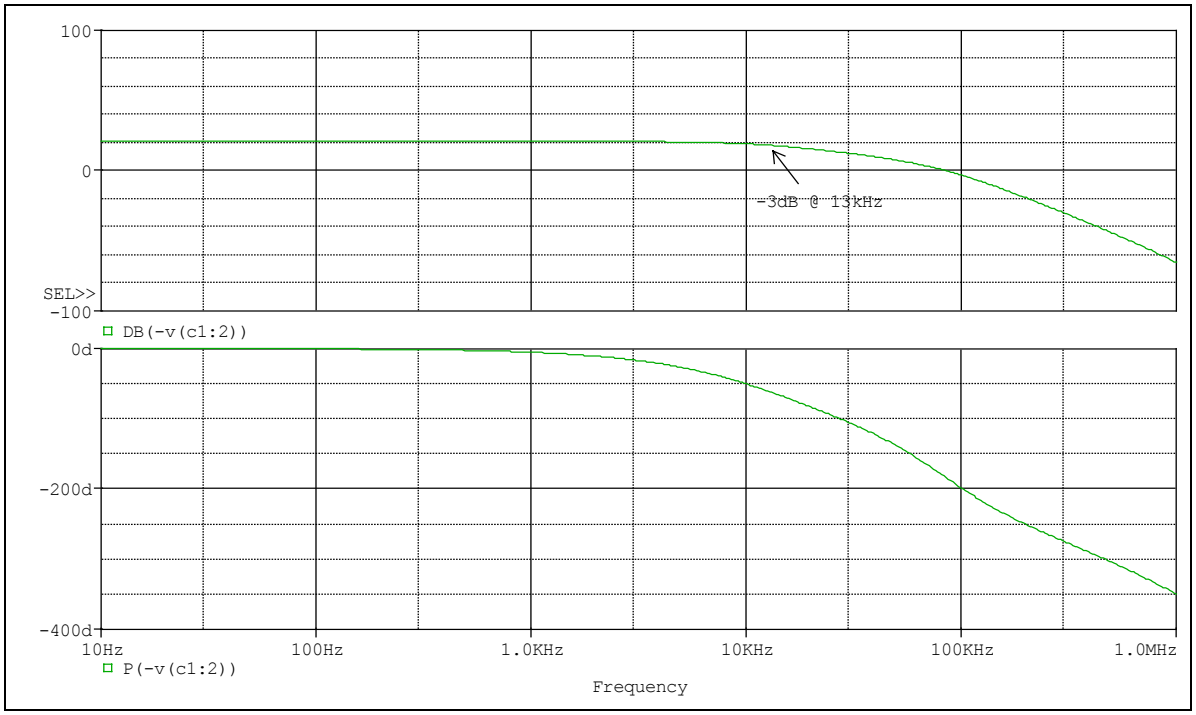


Figure 14 – Bode plot of actuator drive electronics. DTT mirror actuator (10.8µF).

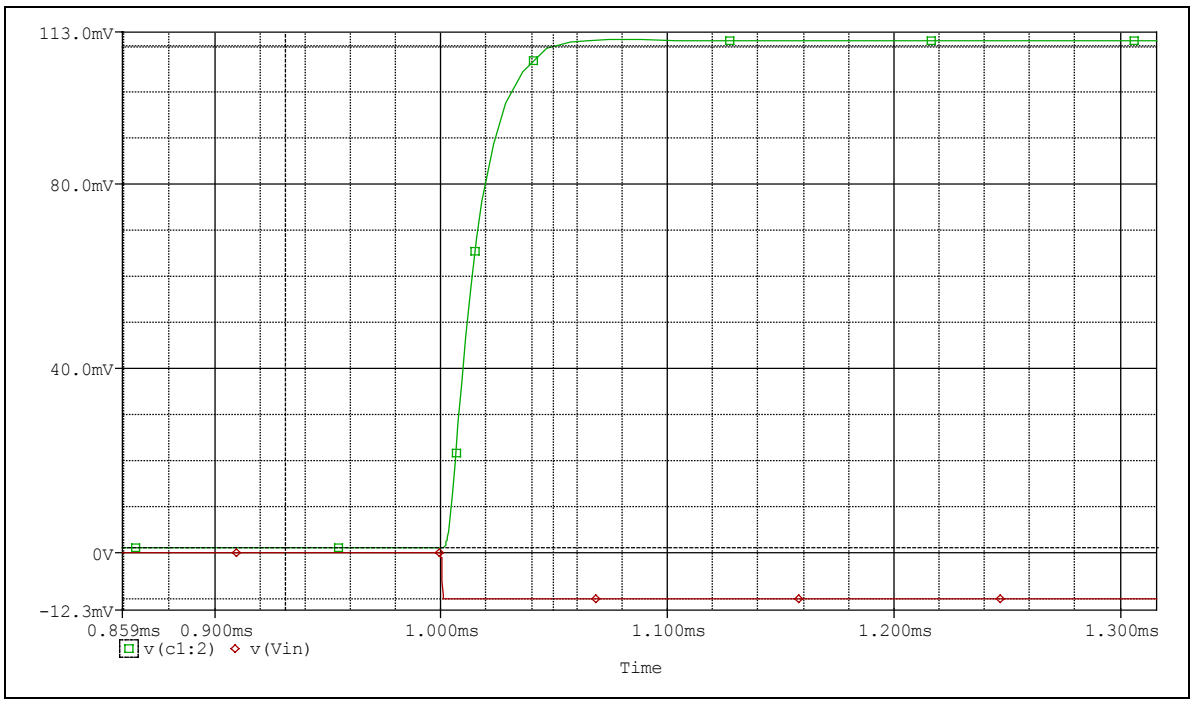


Figure 15 – Step response of actuator drive electronics. DTT mirror actuator (10.8µF).

- *Drive and actuator transfer function.* From the PSPICE simulation, we have derived by a least square algorithm the corresponding continuous transfer function, taking into account the drive dynamics and the capacitive load of the actuator. The transfer function refers to the electrical response only, not including the mechanical system response. The system is well fit by a dominant single pole at 12.7kHz and a complex poles pair at 72kHz.
- *Determination of mirror transfer function.* The transfer function from actuator voltage input to angular displacement has been obtained from the plots reported on page 2-12 of [RD4]. The actual data acquired during the experimental measurement of the transfer function were not available, therefore the transfer function has been manually inserted considering two eigenfrequencies, the first at 700Hz with a damping factor of 0.6, the second at 1kHz with a damping factor of 0.09. For a preliminary evaluation, this 'manual' way of matching the transfer function has proved to be more accurate than a least square method based on the actual transfer function, probably due to the limited number of input points derived from the mirror bode plot.

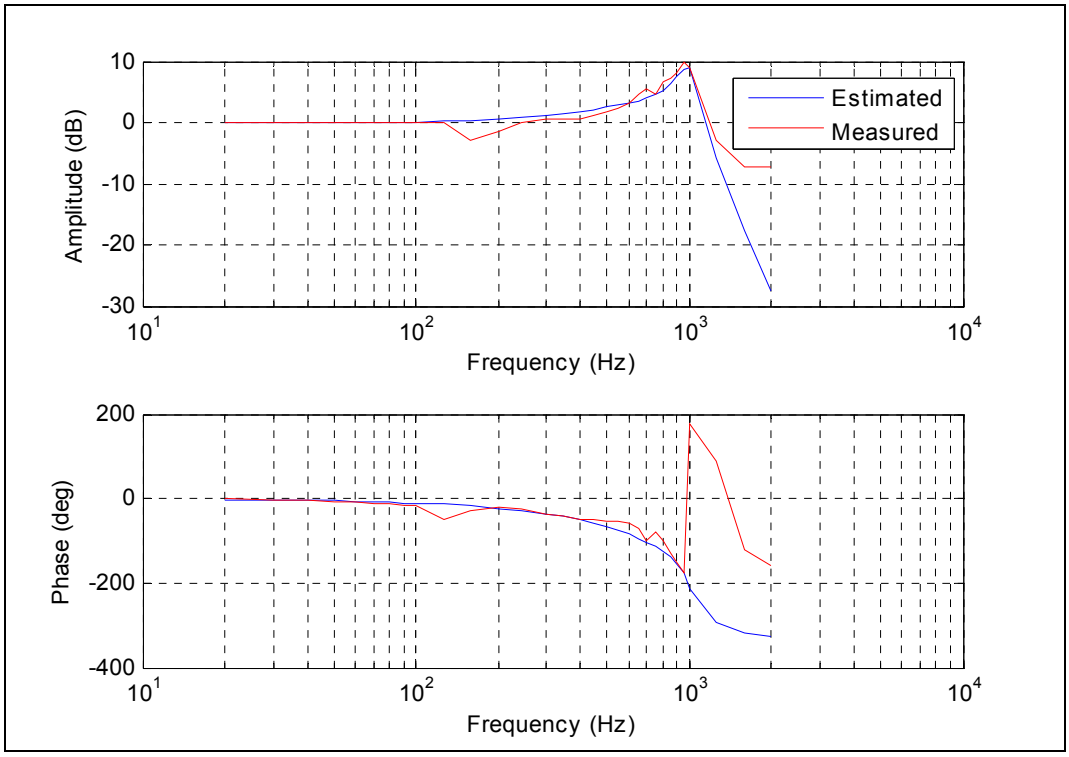


Figure 16 – Estimated vs. actual mirror transfer function.

- *Optimization of dynamic controller.* The open and closed loop transfer functions of the controlled systems have been computed. As compensator, we have initially used a simple PID controller. The control coefficients have been preliminarily optimized by iterative functional minimization. This procedure simulates the procedure we intend to use for automatic coefficient optimization on the real system. The minimization functional is the Integral Square Time^2 Error, applied to a step response. We obtained a **bandwidth of 550Hz** (see Figure 17), with a good phase margin of 65 deg and a quite poor gain margin of 11dB (see Figure 18).

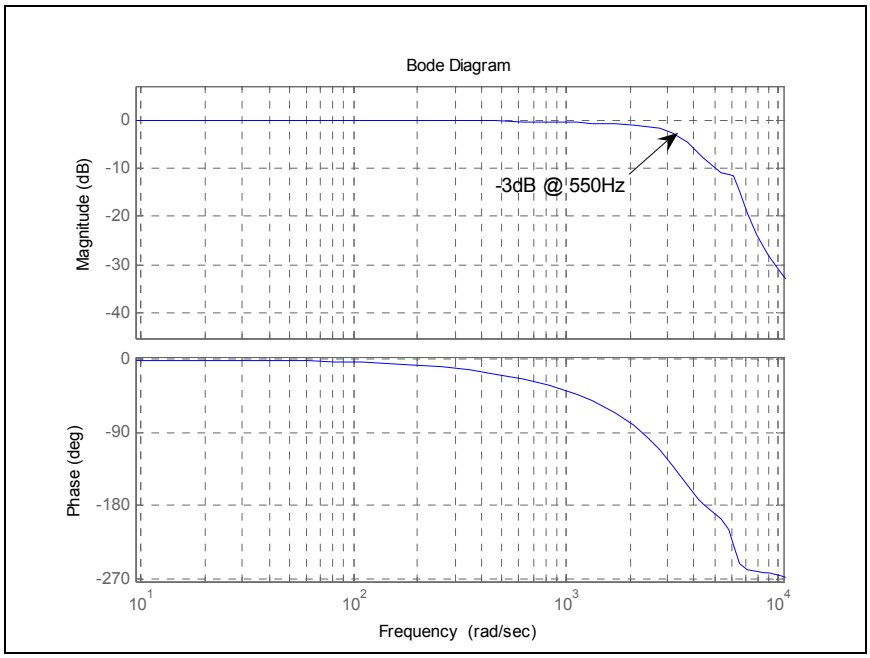


Figure 17 – DTT control. Closed loop transfer function with gain and phase margins.

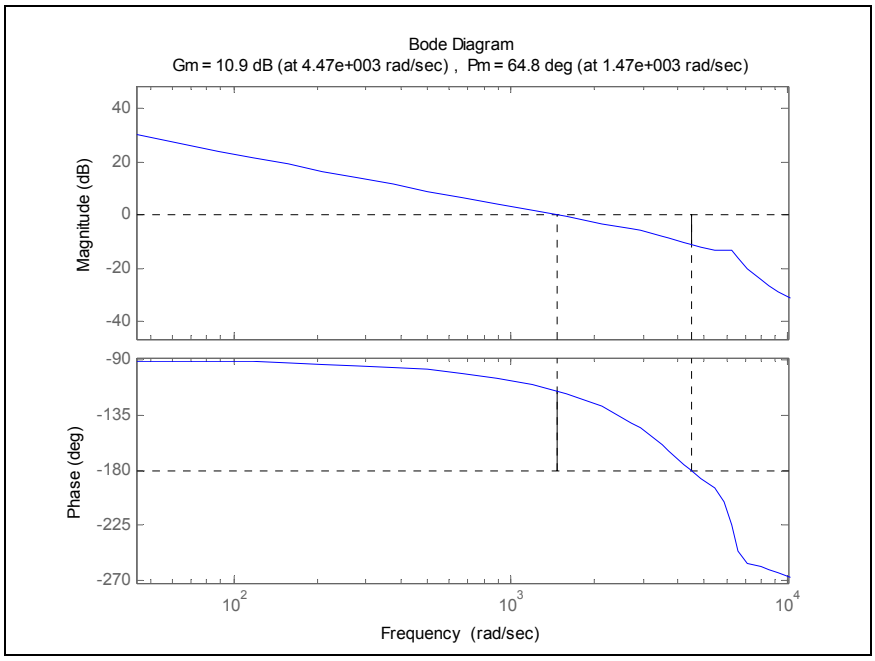


Figure 18 – DTT control. Error transfer function with gain and phase margins.

- *Simulation of system considering non-linear effects.* The closed loop behavior of the system has been simulated using Simulink. At this point, some other non-linear elements have been added to the simulation scheme:

- Digital quantization. It has been assumed that the full stroke motion and full stroke output signals are both divided in 2^{16} steps.
- Computational delay. The computational delay, including ADC sampling, conversion, reading and DAC writing and settling is 7.4μs. This value is extrapolated from actual data measured on similar systems.
- Noise. The noise term generated by the strain gage amplifier is relevant, mainly due to the poor sensitivity of the strain gage installed on the actuator (25 μV/μm with 5V bridge supply, data measured on P840.61 DTT actuator). This forces to amplify the strain gage signal by a factor of 1000, therefore the voltage noise of the first amplifier becomes relevant. Neglecting the effect of external noise sources, like pick-up along the connection to the strain gage, and taking into account the noise sources on amplifiers and resistors (flicker, burst, avalanche), the integrated noise up to 70kHz (CLMP rate), expressed as corresponding on-sky angle, amounts to 13.1 mas RMS. Limiting the noise integration to the specified closed loop bandwidth of 200Hz, the noise is 1.23 mas RMS on-sky.
  Another contribution comes from the high voltage drive noise. At simulation level, this contribution is totally negligible.

The simulation block diagram is presented in Figure 19. Figure 20 reports the results of 1 second of 'bad seeing' activity of the actuator at 0 deg (input data are the same of Figure 11). The 'following error' RMS is **2.8 mas**.

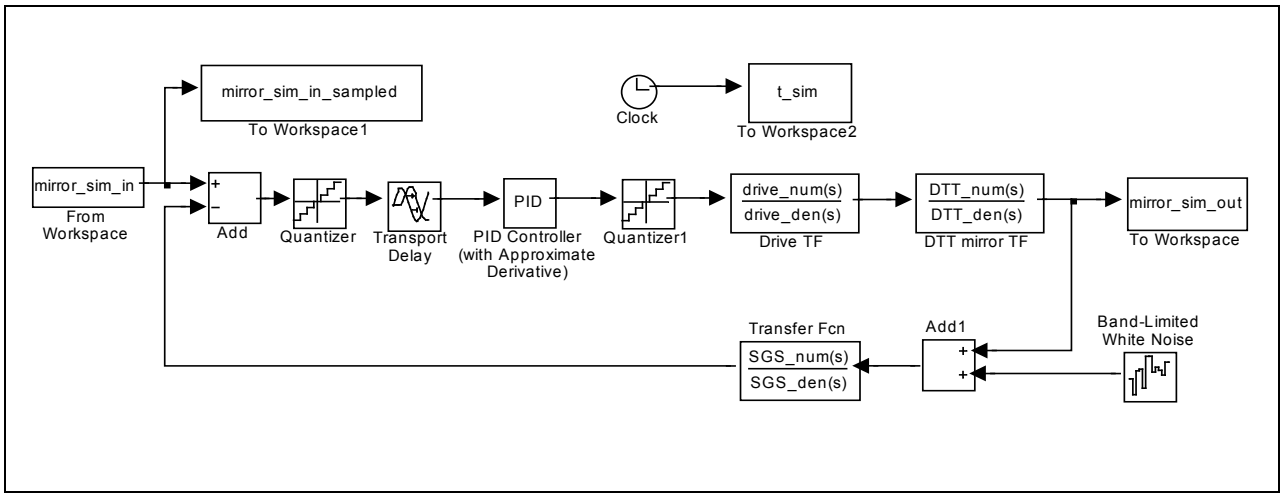


Figure 19 – Simulation block diagram.

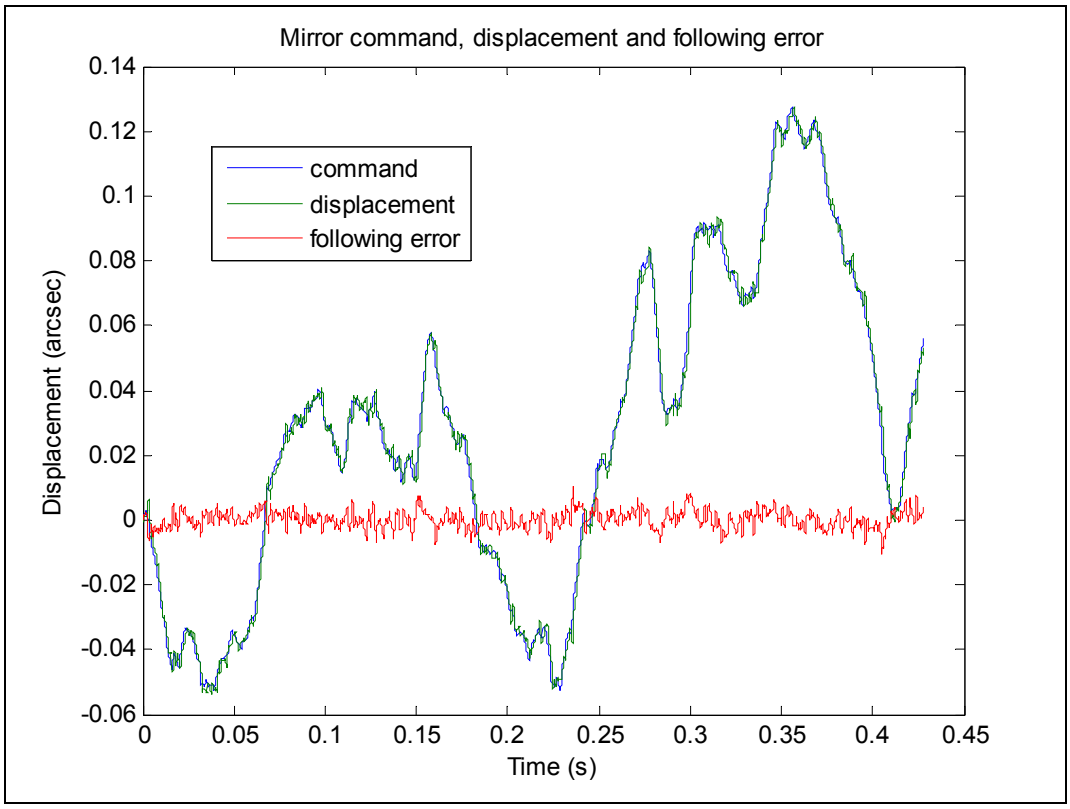


Figure 20 – Simulation of a 'bad seeing' input to actuator at 0 deg.

- *Pspice simulation of 'actual' actuator commands to check for saturation.* The simulation data, taken at drive input, has been used as stimulus for the electronic simulation, to verify that no saturation due to current or voltage limit occurs.

Comments to simulation results.

- The simulation considers the activity of a single actuator, as if the three actuators would be ideally decoupled. This is probably an acceptable simplification, considering the intrinsic high rigidity of the actuators. Their self-resonance is at 6 kHz, while the mirror main resonance is at 1 kHz. This is a significant separation.
- The simulation does not take into account the electrical crosstalk between actuator and strain gauge sensors.

6.1.1.3.5 Experimental tests

In order to validate the above simulation tests and to prove the validity of the described design concept, we have conducted a test campaign on a bread-boarded prototype of the actuator drive.

6.1.1.3.5.1 Test setup

The test setup comprehends the following main components:

- Breadboard of a single channel high voltage drive amplifier. The prototype implements the above described circuit, apart from the 'safety' components (output relays, protection diodes) and monitor features. The high voltage drive supply is set to –35/+135V, as in the final configuration, i.e. we tested also the 'overdrive' capability of the actuator.

- Breadboard of first stage of the strain gage signal conditioner (differential amplifiers).
- Microgate AdOpt DSP board, properly configured to operate as digital part (real-time control) and low voltage signal processor of the HVC board.
- PI840.61 DTT actuator installed on a dedicated mechanical fixture where the pre-loaded actuator drives a mass tailored to reduce the self-resonance frequency down to 1kHz, in order to roughly emulate the dynamic behavior of the DTT mirror.

To maximize the efficiency and reduce the effort to complete the test setup, we used the AdOpt components installed on the internally developed testbench usually dedicated to automated testing of AdOpt systems.

In terms of software, the CLMP has been implemented in an almost final configuration. For the tests we have made extensive use of the '*dynamic buffers*', natively available on the AdOpt DSP boards, allowing us to upload the desired real-time command sequences and to read back the position sampled at full control speed (70kHz).

All tests have been sequenced by means of Matlab scripts, using the MGP communication protocol.

Here following the list of tests and measurements performed:

- Measurement of actuator histeresys (clearly in open loop) over different voltage ranges.
- Open and closed loop transfer function.
- Control loop optimization using automated script.
- Open loop and closed loop step response.
- Response to the same time-history used in simulation (see Figure 20), both in open and closed loop.

6.1.1.3.5.2 Test description and results

IMPORTANT NOTE: during the execution of the tests we experienced a major failure, in fact one branch of the actuator strain gage failed suddenly. Since, at that time, we had not collected final data for this document, we had to recover to this situation operating the actuator with just half of the strain gage bridge (the other half was replaced by two fixed resistors). The performance is very much affected by this reduced configuration, in fact the sensitivity is halved and we see a huge crosstalk between actuator command (or, more correctly, its derivative) and sensor reading. Due to this reason, the results presented hereafter are provisional and cannot be considered to be representative of the final performance. Transfer function tests and control loop optimization are not presented at all, because they are meaningless in these conditions. We are waiting for a replacement of the failed actuator and expect, hopefully, to present the final results during the DDR meeting.

Actuator histeresys

It is well known that piezoelectric actuators exhibit a significant histeresys. We have tested the actuator to verify the entity of such behavior at different voltage spans (30/60V, 10/90V, -20/+115V) and different speeds ($5Vs^{-1}$, $18Vs^{-1}$ and $85V\ s^{-1}$). By evidence the test has been conducted operating the actuator in open loop.

Some of the results are presented in Figure 21 to Figure 24. The histeresys amplitude is remarkable, up to 17µm for the same applied voltage on the –20/+115V test. We noticed a very weak dependence on the slew rate, namely the effect tends to increase reducing the speed. This is also confirmed by the large deviation of the first cycle if the actuator is left on the initial position for long time, as in Figure 24, where the test has been executed after driving the actuator at -20V for 5 minutes.

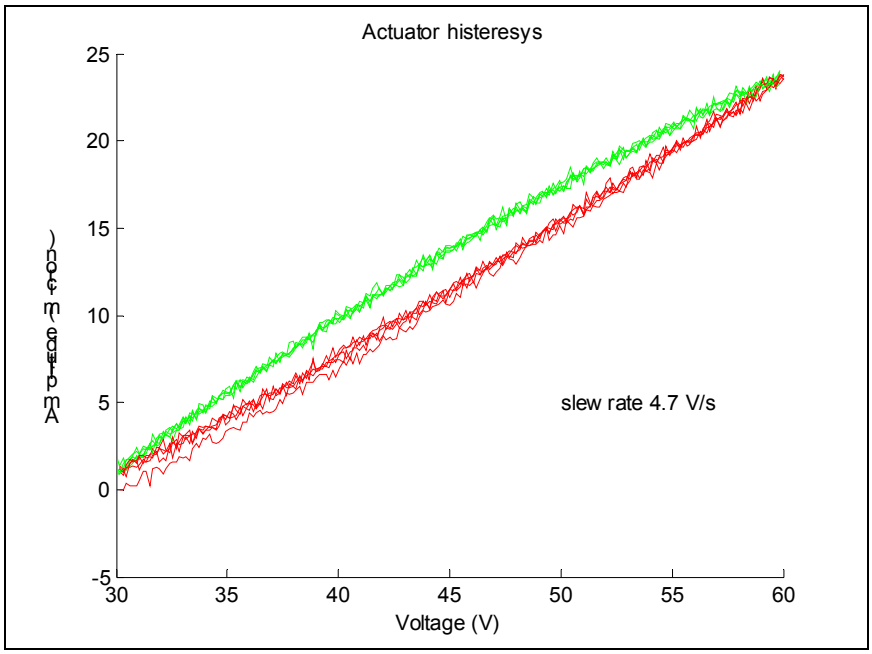


Figure 21 – Actuator histeresys. Voltage span: 30/60V, slew rate 4.7Vs^{-1}

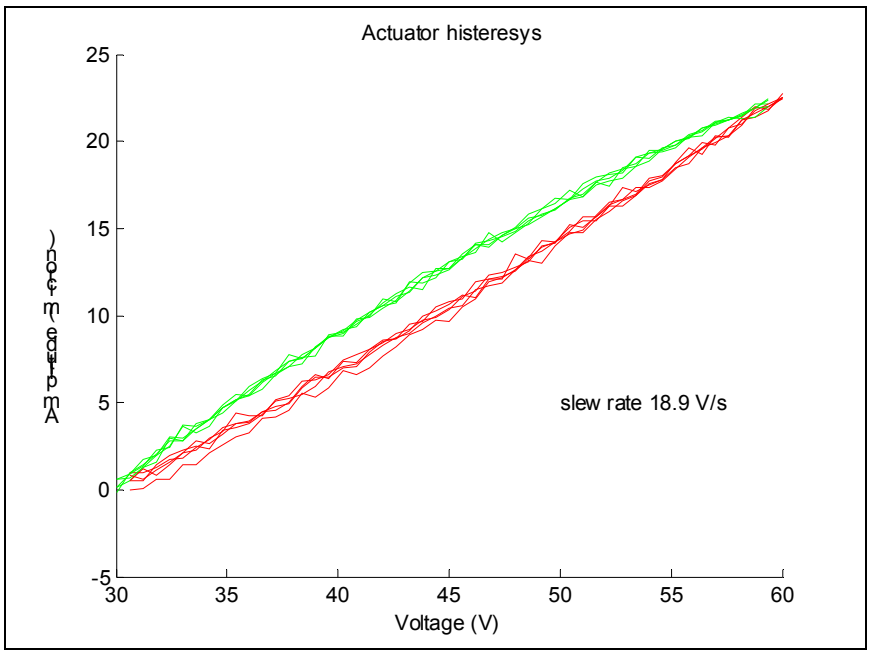


Figure 22 – Actuator histeresys. Voltage span: 30/60V, slew rate 18.9Vs^{-1}

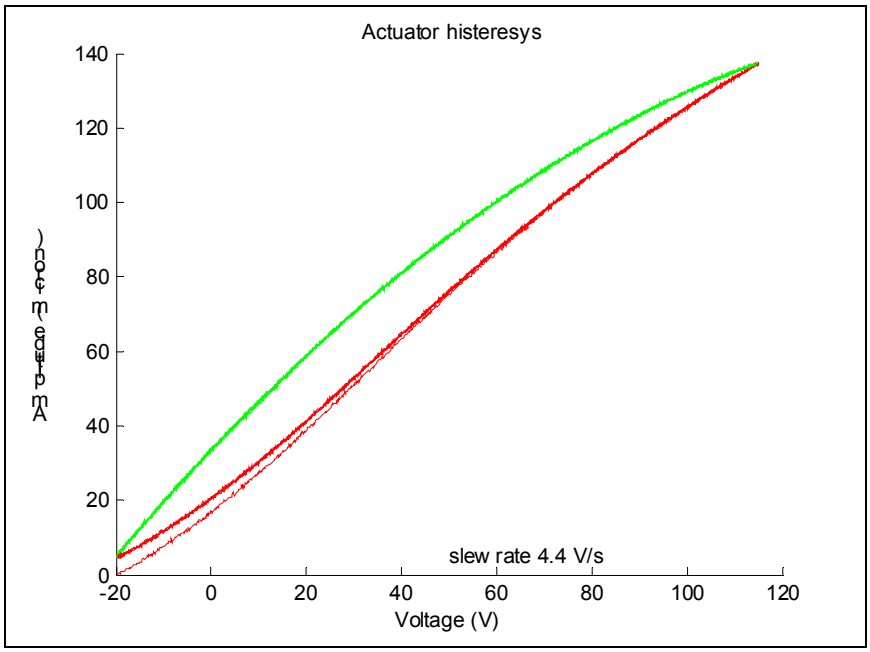


Figure 23 – Actuator histeresys. Voltage span: -20/+115V, slew rate 4.4Vs^{-1}

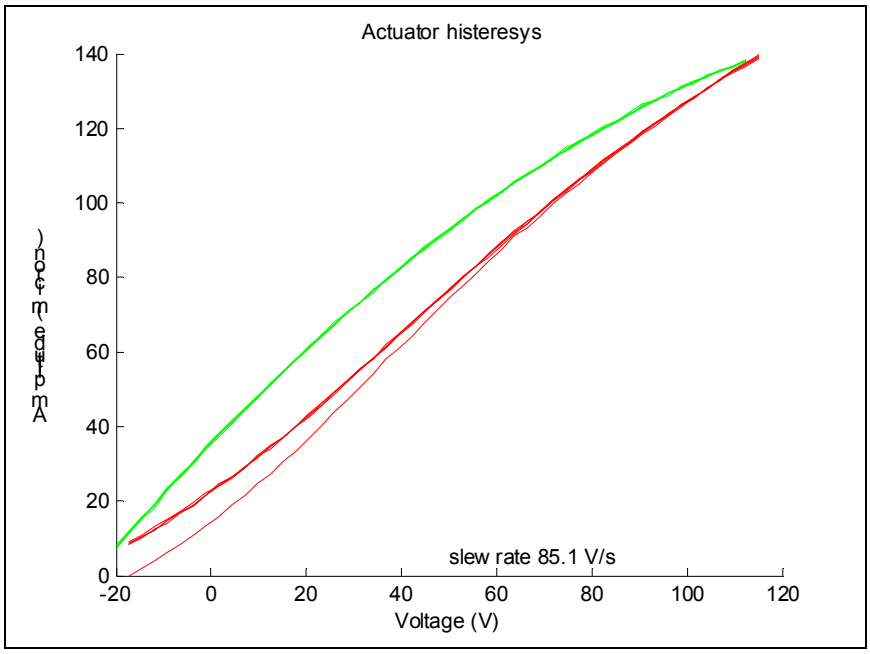


Figure 24 – Actuator histeresys. Voltage span: -20/+115V, slew rate 85.1Vs^{-1}

Step response

We report here the results to a step command with 2µm amplitude. The position output is filtered numerically with a 5th order butterworth filter at 4 kHz. The rationale behind this post-processing is in the fact the position acquired at 70kHz is not the *actual* actuator position; it is affected by the noise of the strain gage and related amplifier. Filtering at 4kHz is still representative of the actuator dynamics, being the mechanical resonance at 1kHz, but reduces the artifacts introduced by the acquisition chain.

The open loop response puts to evidence once more the large static error due to histeresys. The error is particularly evident on a small amplitude step, while on larger movements the actuator tends to behave more linearly, thus reducing the relative error.

The large overshoot is not real; it is an artifact caused by the problem occurred on the actuator, in fact it was not present before it (unfortunately we can not show this because data were not collected). Since one branch of the strain gage bridge is missing, the output is single ended instead of differential, and therefore the rejection to the actuator command voltage derivative is very poor.

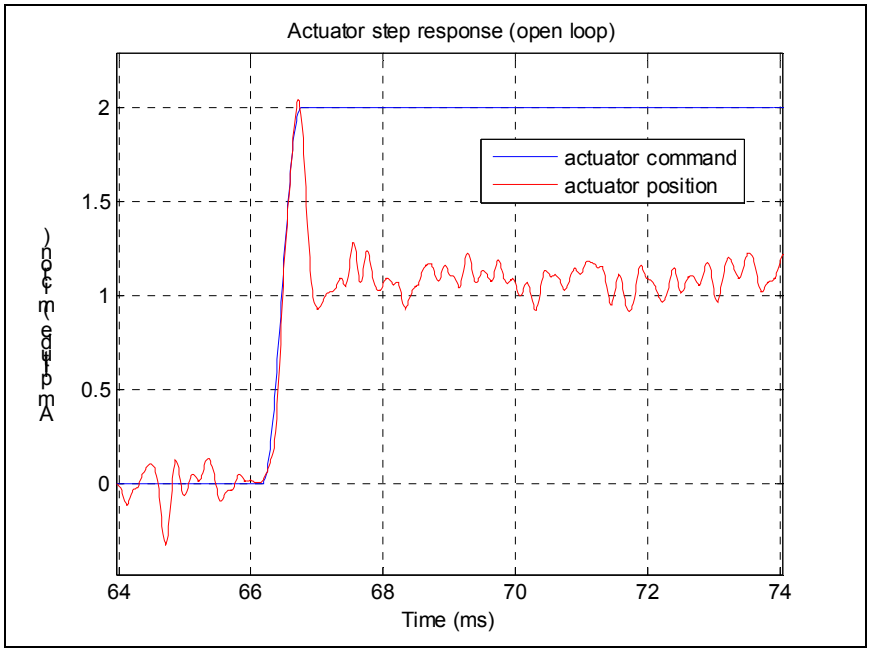


Figure 25 – Open loop step response, 2µm commanded step. The overshoot is mainly an artifact due to crosstalk.

The closed loop response shows that the static error goes to zero thanks to the integral part of the dynamic controller. The effect of the histeresys is evident from the large difference between actuator command and actuator position.

As already stated, the control law adopted during this test is very preliminary and not optimized. The law implements a simple PI (Proportional-Integral) controller. Also in this case the effect of crosstalk is quite evident, nevertheless the feedback is able to react containing the overshoot.

Extrapolating the bandwidth from the step response, we measure a characteristic time of ~0.7ms (an accurate measurement is not possible due to the crosstalk disturbance), which corresponds to ~230Hz bandwidth. Even in this non-optimal condition, it is within specification.

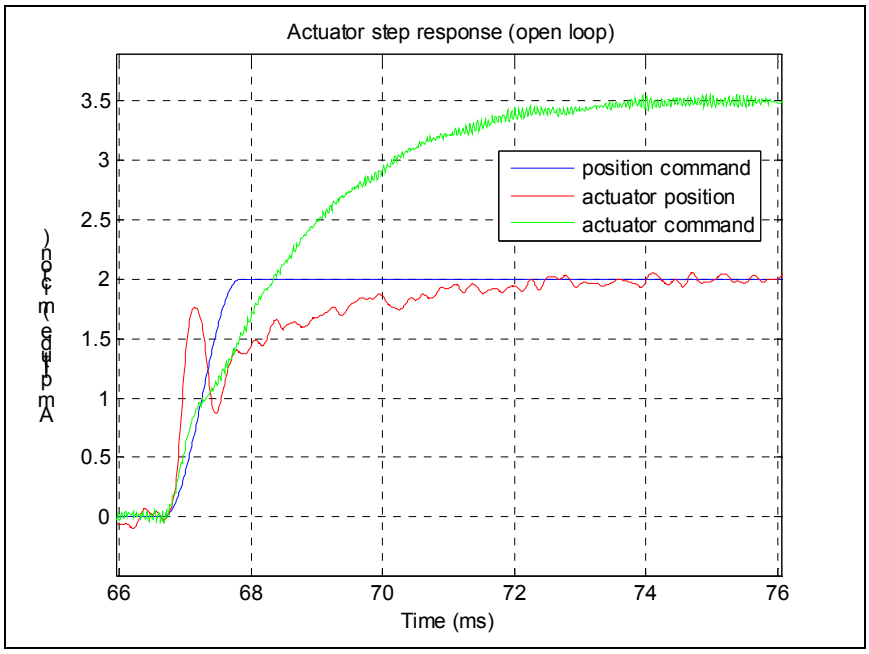


Figure 26 – Closed loop step response, 2µm commanded step.

Time-history response.

The time-history test gives the most realistic perception of the drive behavior in real environment. To this aim, we used the same time-history used for simulation (see §6.1.1.3.4 and Figure 20). It comes from real data acquired during observation in 'bad' seeing conditions.

The test has been performed over 0.43s of observing time. The time history is input at 1kHz, using Zero-Order Hold (ZOH) approximation to oversample it to the control loop rate of 70kHz.

The open loop results are presented in Figure 27, while Figure 28 reports the closed loop ones. The following error is reduced by a factor 5.5.

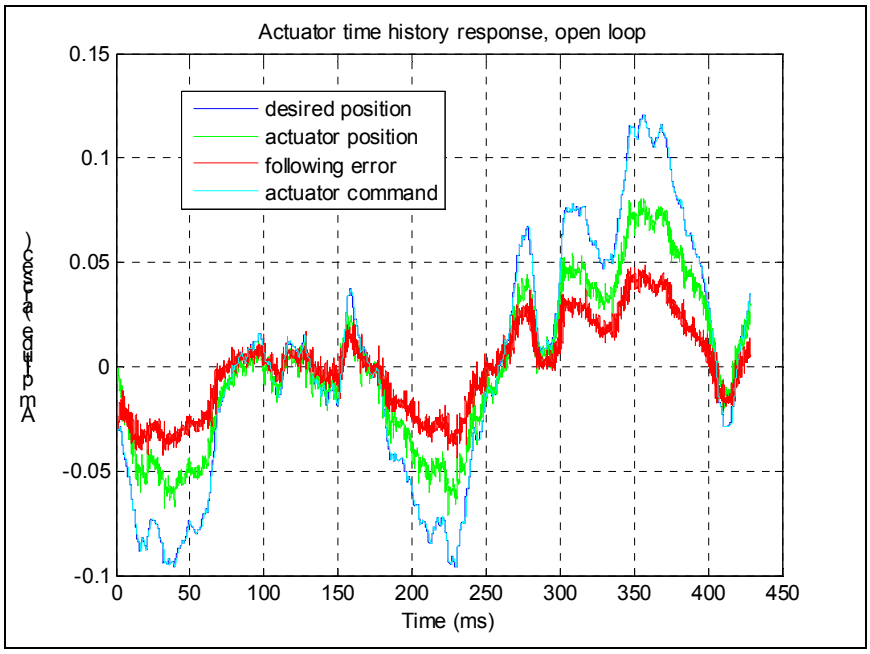


Figure 27 – Time history test in open loop. Following error: 1.69μm RMS corresponding to 21.2 mas RMS on-sky.

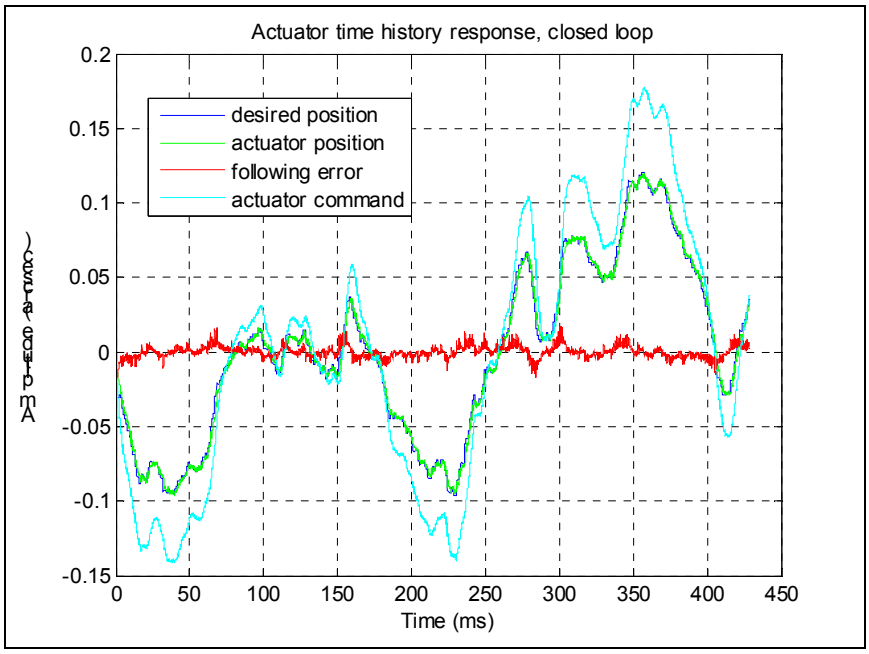


Figure 28 – Time history test in closed loop. Following error: 0.31μm RMS corresponding to 3.86 mas RMS on-sky.

At a first impression, the open loop response seems to be simply affected by a gain error. In reality this is not true, in fact the gain is the correct one but, as already pointed out in the step test and confirmed by the histeresys analysis, on small movements the histeresys affects the gain significantly.

Second, one might argue that the open loop results are not representative of the real behavior, in fact the outer optical loop would compensate for a large amount of the following error. This is true, but it should be also considered that the optical loop bandwidth might be poor, especially in low flux and bad seeing conditions, thus reducing the possibility of effectively correcting the error. This just a qualitative comment; to quantify this aspect, the reported results should be used as input for the error budget and/or optical loop simulation.

General comment on experimental results.

The presented results, even if strongly limited by the actuator failure, allow us to conclude that the adopted design is suitable to reach the desired performance. There is some concern on the actuator noise due to the signal conditioning chain, but it is actually difficult to do better, significant improvements on the circuit are not easy because it is already quite well optimize in terms of noise. We will investigate with PI (the actuator manufacturer) if the strain gage can be supplied with higher voltage (we are currently using 5V, most like 10V should be also acceptable), thus increasing its output gain and allowing to reduce the amplifier gain.

Nevertheless, the following error is already acceptable (even with the broken actuator) and it is more limited by dynamics rather than by noise.

We are confident that the dynamic performance can be improved furthermore as soon as the replacement actuator will be available.

6.1.1.4 Backplane

The system backplane is a passive board with the card-edge connectors for the AdOpt boards. There are different versions of backplane, with up to 17 available slots. For the NGWFC MGAOS we have foreseen a 12 slots backplane, that assures significant expansion possibility and allows at same time to fit all components (MGAOS and VME components) on a single 19" crate.

The backplane is completely passive. It comprehends two busses, 16 bit wide for diagnostic and 32 bits wide for real-time. Both busses operate at ~60 MHz, DDR. Considering the high communication speed and in particular the heavy load on the bus (the loaded bus impedance is down to 23Ω), we have selected B-LVDS technology. Thanks to this design choice, the backplane works properly regardless of the number and position of installed boards. Another advantage related to the use of B-LVDS technology is the reduction of EMI-RFI emission and sensitivity to external disturbance.

Each slot on the backplane is uniquely identified by hardware signals. In this way, it is not required to configure the AdOpt boards in any way. The correct addressing of the board is performed at system startup by an automatic procedure that recognizes the position of the board with respect to the other installed boards.

The backplane distributes also all power supplies (3.3V, ±12V, High Voltage) and some hardware flag dedicated to system operation (see Table 3 and Table 4).

While designing the system, great importance has been given to the reliability of the backplane, in particular considering the large number of connection points. To this aim, all connections are doubled on the two sides of the card edge connector. High quality, gold plated connectors are used and the gold plating thickness on the board terminals is checked during PCB production. To reduce the mechanical stress on the backplane during boards insertion and extraction, the backplane PCB is 3.2mm thick.

6.1.1.5 WFS to BCU and BCU to DM HVA interface boards

The WFS and the DM HVA are interfaced to the PIO ports on BCU (see §6.1.1.1.11) by means of interface boards. These simply provide level shifting (from RS644 to LVTTL for WFS AIA interface and from LVTTL to RS485 for DM HVA interface) and proper connectors according to the existing WFS and DM

HVA interfaces (see §6.3.2.3). The BCU to DM HVA interface board also provides the interface for STRAP synchronous operation and to IRIG board for system time-stamping synchronization .

The WFS to BCU interface comprehends also the serial line used for SciMeasure CCD controller configuration. The BCU acts as bridge between WIF (where the CCD configuration actually occurs) and the Little Joe controller.

Mechanically both boards are mounted on a single slot MGAOS front panel, see Figure 2.

6.1.2 VME components

The VME crate is based on a Motorola Power-PC single board computer.

The other main components on the VME crate are:

- Symmetricom ttm635VME timing board
- STRAP (provided by Keck)
- Reset Board
- Power supply unit

The MGAOS is also integrated into the VME crate and is described separately in 6.1.1.

6.1.2.1 CPU board

The CPU board is the high performance MVME6100-0173 Power-PC VME single board computer from Motorola.

The key features of the MVME6100 board in the selected configuration are reported hereafter:

- Processor: MPC7457 PowerPC® processor running at 1.267 GHz,
- Co-Processor: 128-bit AltiVec™ coprocessor for parallel processing
- RAM Memory: 1GB DDR ECC memory + 512KB L2 cache + 3MB L3 cache
- Permanent memory: 128 MB Flash
- On board expansions: two PMC expansion slot
- VME bus interface: 2eSST VMEbus protocol with 320MB/s transfer rate across the VMEbus
- Ethernet: two Gigabit ports over copper
- Serial: two 16550 compatible ports
-

6.1.2.2 IRIG decoder

To synchronize the telemetry time-stamping clock with an external timing source the RTC is equipped with the Symmetricom ttm635VME timing board

The main characteristics and functionalities of this board are the following:

- Time on demand (days through 0.1 microseconds) with zero latency. This feature is implemented with hardware registers which latch the current time upon host request.
- Event logging (days through 0.1 microseconds). This feature is implemented with a second set of hardware registers. Time is captured on a positive or negative input edge.
- Six operational modes are supported. Modes are distinguished by the reference source.

Mode	Source Of Synchronization
0	Timecode – IRIG-A IRIG-B XR3 2137 NASA 36
1	Free running - on board VCXO used as reference.
2	1 PPS - accepts input one pulse per second.
3	RTC - uses battery backed on board real time clock IC.
5	GPS (optional) - double wide configuration including GPS receiver.
6	GPS (optional) - uses GPS receiver/antenna (receiver in antenna).

- Provides an output clock synchronized to the selected reference; programmable 1, 5, or 10MHz TTL.
- All modes of operation are supplemented by flywheel operation. For example, if synchronization source is lost, the TFP will continue to function at the last known reference rate.
- Generates synchronized IRIG B timecode. Modulated and DC level shift formats are produced simultaneously. Also generates IRIG H DC level shift.
- Programmable frequency output (periodics) is provided. The output frequency is 10,000,000/ (n1 * n2), where 1<n1<65536 & 1<n2<65536.
- A time coincidence strobe output is provided. Programmable from hours through milliseconds. This strobe also has an each second mode programmable to milliseconds.
- Five maskable interrupt sources are supported. IRQ levels one through seven are programmable.
-

Int.	Source Of Interrupt
0	External event input has occurred.
1	Periodic output has occurred.
2	Time coincidence strobe has occurred.
3	One second epoch (1PPS output) has occurred.
4	Output data packet is available.

-
- Time-of-day, hours, minutes, and seconds are displayed on front panel LED's.
- Most inputs and outputs are accessible via the P2 connector.

6.1.2.3 Reset board

The reset board handles the external reset closure and generates a ‘clean’ reset pulse for both the VME bus and the MGAOS system. The reset input signal can be totally galvanically isolated from the system, or it can be referenced to ground by the user, inserting a jumper.

The reset board is mounted on the rear part of the VME backplane, directly connected to P1. The power supply is directly taken from VME.

A reset on VME always generates a reset of MGAOS.

6.1.3 Power dissipation

The total power dissipation is reported on Table 10. The reported values refer to

Device	Dissipation
MGAOS BCU	12.5 W
MGAOS DSP (3x)	20.4 W
MGAOS HVC	13.7 W
VME CPU BOARD	42 W
VME IRIG BOARD	8.5 W
STRAP	19.5 W
Total device power	*116.6*
Power conversion efficiency	75%
Total dissipated power	**155 W**

Table 10 – System crate power dissipation summary

The typical power supplied by the MGAOS and VME PSU is 108W, while the HV PSU supplies just 8.6W.

6.1.4 Power supply system

The RTC power supply comes from two units, one supplying the VME and MGAOS system the other supplies the HVC. Both power supplies are installed in the rear of VME crate.

6.1.4.1 VME and MGAOS power supply

The MGAOS requires +3.3V, ±12V and high voltage supply. VME requires +5V and ±12V. All these power rails (with exception of the high voltage) are available on quad-output PSUs typically used for Compact PCI systems. Therefore, even if the typical VME PSUs are triple output, we decided to use a Compact PCI PSUto reduce the number of parts and allow simultaneous power up of the different subsystems.

The PSU will supply the following components

- Motorola CPU board
- Timing board
- STRAP
- MGAOS

The preliminary choice is Schroff CPCI-AC-6U-400.

This power supply unit has an overall load capability at sea level of 400 W. Applying the de-rating for high altitude operation, the maximum output power is reduced to

400 x 0.8 = 320 W

The voltages and maximal currents are listed in the following table

Voltage	available current (max)	MGAOS	Motorola CPU	Irig Timing board	STRAP	TOT current
+5V	50A		10.2A	1.5A	5A	16.7A
+3.3V	80A	10A				7°
+12V	7.5A	0.060 A		0.050A	1.5A	1.61A
-12V	1.5A	0.060 A		0.030A	1A	1.09A
					TOT Power (max)	**149W**

Table 11 – Voltages and currents of RTC power supply rails.

The power supply is well oversized with respect to the actual power demand.

6.1.4.2 HVC supply

The HVC power supply requirements are listed hereafter:

Requirement	Value	Comment
Output voltage, positive rail	>132V	120V maximum actuator voltage + 12V for power amplifier headroom
Output voltage, negative rail	<-32V	-20V maximum negative actuator voltage - 12V for power amplifier headroom
Output current	> 120mA x 6 = 720mA	Specification based on maximum amplifier peak current. It exceeds significantly the expected maximum current on each channel. Conservative assumption.
Availability of over-current protection	-	-

Table 12 – High voltage power supply unit requirements.

The selected power supply configuration comprehends three Lambda type ZUP 80-2.5 power supplies, two of which in series for the positive supply and one on the negative rail.

The main characteristics of this power supply are reported in Table 13.

Specification	Value
Input voltage	85~265 Vac, 47~63 Hz
Output voltage	Adjustable, 0~80 V
Output current	0~2.5 A (adjustable limit)

Table 13 – Lambda ZUP 80-2.5 specifications.

This power supply unit can be remotely controlled through a built-in RS485 interface. Up to 32 units can be connected on the same multi-drop serial line. Alternatively, an interface module (model GP485) provides conversion from GPIB to RS485. Also in this case one single GPIB connection allows controlling up to 32 power supply units. These power units will be controlled by a GPIB interface provided by Keck, similar to the one that already controls the HVA power supply. Anyway, the control of these power rails is not particularly critical in terms of system safety, in fact the HVC boards have their own safe start-up and shutdown procedures (see §7.1.5.4).

6.1.5 Mechanical aspects and cooling

The MGAOS is contained in a dedicated chassis that fits into a standard 6U, 19”crate (see Figure 2). The chassis appears as a ‘sub-crate’ with 12 slots, 14mm spacing. The width of the chassis is 223.52mm, i.e. 44 TE. The chassis does not make use of standard VME backplane. All power and control connections required for system operation are on standard DIN 41612 connector on the sub-crate rear panel.

6.1.5.1 Cooling

The system is air-cooled and forced air is provided by a standard fan-drawer, that also cools the VME components installed on the main crate.

Considering the low power dissipation (see Table 10), and also on base of previous experience, a forced air flow of 100 LFpM. The required air flow has to be increased according to pressure reduction due to altitude. Therefore a flow of 100/0.8 = 125 LFpM shall be obtained. This can be easily achieved by a standard 19” fan drawer. We plan to install two 3-fans, 1U drawers, one on the front side of the crate (cooling VME boards and MGAOS), the other on the rear side, cooling the PSUs.

We plan to measure the actual air speed on components in order to verify experimentally that the above specification is actually met.

6.2 TRS Hardware

The TRS Hardware is based on two main components:

- Storage Server
- Disk Array

The storage server is linked to the MGAOS trough a FibreChannel link and to the LAN through Ethernet.

The Disk Array is connected only to the Storage Server through a FibreChannel link. Both FibreChannel links are configured as point-to-point topologies. The interface between the Server and the Disk Array supports both Arbitrated Loop and Fabric Switching topologies; it also supports FC-4 Layer SCSI-FCP and FC-0 Layer Hot Pluggable SFP. Thus an upgrade to a SAN system is straightforward.

6.2.1 Storage Server

The storage server is an SUN X4100 Model. This is a high performance 19” 1U rack type server with this main characteristics:

- CPU: Dual AMD Opteron Model 252 Processors
- 2-GB Memory
- Disk: 2x36-GB 10000 RPM SAS Disk Drive
- Redundant Fans
- 2 Power Supply Units
- 4 10/100/1000 Ethernet Ports
- FibreChannel Device: 2 SANblade Qlogic 2340

Just the operating system and the query/storage software resides on the server. All telemetry data is stored on the disk array.

6.2.2 Disk array

For the disk array system the SurfRAID TRITON 16FA has been chosen. This is a SATA RAID controller with FibreChannel interface. It is equipped with 16 Hitachi 400GB SATA disks for a total raw storage capacity of 6 TB. This is slightly more then the real storage needs, but this allows to configure the system with some redundancy. In fact the TRITON 16 FA supports RAID levels 0, 1, 3, 5, 6 and 0+1.

The basic idea of RAID is to combine multiple disk drives into an array of disk drives to obtain performance, capacity and reliability that exceeds that of a single large drive. The array of drives appears to the host computer as a single logical drive. Each RAID level provides disk fault-tolerance with different compromises in features and performance. Here are a brief description of the various RAID levels:

- RAID 0 is defined as a group of striped disk drives without parity or data redundancy. RAID0 deliver the best data storage efficiency and performance of any array type. The disadvantage is that if one drive in a RAID 0 array fails, the entire array fails.
- RAID 1, also known as disk mirroring, is simply a pair of disk drives that store duplicate data but appear to the host as a single drive. Each write operation must go to both drives of a mirrored pair so write performance is not increased in respect to a single drive. However, each individual drive can perform simultaneous independent read operations doubling in fact the read performance.

- RAID 3 stripes data across groups of drives, but one drive in the group is dedicated to storing parity information. Records typically span all drives, which optimizes the disk transfer rate. Because each I/O request accesses every drive in the array, RAID3 arrays can satisfy only one I/O request at a time. RAID 3 delivers the best performance for single-user, single-tasking environments with long records.
- RAID 5. In this case parity information is distributed across all the drives. Since there is no dedicated parity drive, all drives contain data and parity, read operations can be overlapped on every drive in the array. Write operations will typically access one data drive and one parity drive, however, because different records store their parity on different drives, write operations can usually be overlapped.
- RAID 6 is similar to RAID 5 in that data protection is achieved by writing parity information to the physical drives in the array. With RAID6, however, two sets of parity data are used. The main advantages of RAID 6 is high data availability, any two drives can fail without loss of critical data.

Considering that the storage is not shared but is used only by one single host (the TRS server) RAID 3 is probably the best choice achieving best performance with acceptable reliability. To increase availability, one or more drives will be configured as hot spare drives, in case of failure this will automatically replace the damaged disk. The damaged disk can then be hot swapped without interrupting the disk array operation

Using RAID 3 with 2 hot swappable disks the total available capacity amounts to 5.6 TB. The actual storage capability in terms of observing time clearly depends on the observing mode and the amount of configuration data, which is not predictable. Under realistic assumptions we can predict a storage depth of more then five nights (~50 hours) operating at 2kHz frame rate with CCID56.

Another remarkable feature of the TRITON 16FA is that it comes with an environment controller which is capable of accurately monitoring the internal environment such as power supplies, fans, temperatures and voltages. Any malfunctioning can be automatically emailed. Maintenance and configuration of the disk array can be accomplished over the front panel, via a VT100 terminal attached to its serial line, or over Ethernet by telnet or http (web interface) protocol.

The TritonF16 disk array is housed into a standard 19”, 3U cage

6.3 HW Interfaces

In this section we describe the hardware interfaces, distinguishing between *internal* and *external* interfaces. As *internal interfaces* we refer to the connection points connecting internal subsystems of the NGWFC RTC that are delivered by Microgate.

Conversely, *external interfaces* are all connections among Microgate-furnished items and external systems.

Moreover, the hardware interfaces are divided into *electrical* and *data*.

6.3.1 Internal HW interfaces

6.3.1.1 Power and logic interfaces

Interface description	Node A	Node B	Voltage/power rating/level	Connector
MGAOS 'sub-crate' supply and control connector.	MGAOS 'sub-crate'	MGAOS and HV PSU, VME reset board	See Table 15.	DIN 41612 type H7/F24 connector (31 poles)
STRAP synchronization	MGAOS BCU board (PIO #0 port)	STRAP Ext.Sync. input	TTL level. Triggers on falling edge	SMB
Periodic synchronization of RTC	MGAOS BCU board (PIO #0 port)	IRIG board	TTL level. Triggers on raising edge	SMB on MGAOS, BNC on Symmetricon IRIG board

Table 14 – Internal HW interfaces. Power and logic.

Pin	Description	Connects to	Specification
H1,H2, H3	Logic supply	VME+MGAOS PSU	3.3 V, 10 A
F1	Analog supply, positive rail	VME+MGAOS PSU	+12 V, 0.2 A
F2	Analog supply, negative rail	VME+MGAOS PSU	-12 V, 0.2 A
F9	High voltage supply, positive rail	HV PSU	132 V, 0.72 A
F10	High voltage supply, negative rail	HV PSU	-32 V, 0.72 A
H4, H5, H6	Logic ground return	VME+MGAOS PSU	-
H7	Analog ground return	VME+MGAOS PSU	-
F11	High voltage return	HV PSU	-
F12	Logic supply sense+	VME+MGAOS PSU	-
F13	Logic supply sense-	VME+MGAOS PSU	-
F20	MGAOS reset	VME reset board	TTL-compatible, active high
F21	MGAOS booting mode	Available, but not connected	TTL-compatible, high=user, low=default

Table 15 – MGAOS supply and control connector pinout and specifications. Non listed pins are reserved.

6.3.1.2 Communication interfaces

Interface description	Node A	Node B	Connector	Cable
Gbit Ethernet interface between MGAOS BCU and VME CPU board	MGAOS BCU	VME CPU	Standard RJ45 connector	UTP 5E
FibreChannel interface between MGAOS BCU and TRS server	MGAOS BCU	TRS server	2x Dual LC-type connector	2 multimode fibers 62.5-125µm or 50-125µm (**TBD**), 850nm (2.125 Gbit/s, TX/RX)
FibreChannel interface between TRS server and Disk Array	TRS server	Disk Array	2x Dual LC-type connector	2 multimode fibers 62.5-125µm or 50-125µm (**TBD**), 850nm (2.125 Gbit/s, TX/RX)

Table 16 – Internal HW interfaces. Communication.

6.3.2 External HW interfaces

6.3.2.1 Power interfaces

Interface description	From	To	Voltage and power rating	Connector
Crate main supply (unique for VME, MGAOS and HVC PSU)	RTC crate	Main supply	86-264 Vac, 47~63 Hz, max 400 VA	Standard 3-way socket, with EMI-RFI filter.
TRS server main supply	TRS server	Main supply	86-264 Vac, 47~63 Hz, max **TBD** VA	Standard 3-way socket, with EMI-RFI filter.
Disk Array main supply	Disk Array	Main supply	86-264 Vac, 47~63 Hz, max **TBD** VA	Standard 3-way socket, with EMI-RFI filter.

Table 17 – External HW interfaces. Power.

6.3.2.2 Communication interfaces

Interface description	Node A	Node B	Connector	Cable
Gbit Ethernet interface on VME CPU	VME CPU	Ethernet LAN	Standard RJ45 connector	UTP 5E
Gbit Ehternet interface on TRS	TRS	Ethernet LAN	Standard RJ45 connector	UTP 5E

Table 18 – External HW interfaces. Data.

6.3.2.3 Logic and analog interfaces

Interface description	Node A	Node B	Logic level	Connector	Cable
External reset	Telescope control system (TBC)	VME crate	5V to 24V. Active high. Can be isolated from system ground.	SMB (TBC)	Two poles
Interface between SciMeasure Little Joe WFS and MGAOS	SciMeasure Little Joe AIA port	RS644/LVTTL converter on MGAOS	RS644 differential signals	SCSI-type 68 pin connector	Standard SCSI cable (twisted pairs)
Interface between MGAOS and Xinetics HVA	LVTTL /RS485 converter on MGAOS	Xinetics High Voltage Amplifier	RS485 differential signals	96 pole DIN 41612 connector (identical to current interface)	96 poles flat cable (already in use at Keck)
Actuator interface between HVC board and DTT mirror	HVC board	DTT mirror actuators	High voltage control signals and strain-gauge signals	See Table 20.	
Actuator interface between HVC board and UTT mirror	HVC board	UTT mirror	High voltage control signals and strain-gauge signals	See Table 21.	

Table 19 – External HW interfaces. Logic.

	HVC board		Mirror	
Description	**Connector**	**Pin**	**Connector**	**Pin**
HV actuator #A	LEMO EGG.00.302.CLL	1	LEMO FFS.00.250.CTA	1
HV GND, actuator #A		5		2
HV, actuator #B	LEMO EGG.00.302.CLL	2	LEMO FFS.00.250.CTA	1
HV GND, actuator #B		5		2
HV, actuator #C	LEMO EGG.00.302.CLL	3	LEMO FFS.00.250.CTA	1
HV GND, actuator #C		5		2
Strain gauge supply, act. #A	LEMO EGG.00.304.CLL	1	LEMO FFA.0S.304 male	1
Strain gauge GND, act. #A		4		4
Strain gauge +, act. #A		3		3
Strain gauge -, act. #A		2		2
Strain gauge supply, act. #B	LEMO EGG.00.304.CL	1	LEMO FFA.0S.304 male	1
Strain gauge GND, act. #B		4		4
Strain gauge +, act. #B		3		3
Strain gauge -, act. #B		2		2
Strain gauge supply, act. #C	LEMO EGG.00.304.CL	1	LEMO FFA.0S.304 male	1
Strain gauge GND, act. #C		4		4
Strain gauge +, act. #C		3		3
Strain gauge -, act. #C		2		2

Table 20 – DTT mirror interface. The strain gage connection on the HVC board uses a different connector due to space constrains. To connect the actuator, we will use a dedicated cable adaptor with 1-to-1 connection to ‘00’ and ‘0S’ connectors, or, preferably, the connector on the actuator end will be replaced with a ‘00’ series.

	HVC board		Mirror	
Description	**Connector**	**Pin**	**Connector**	**Pin**
HV actuator, X axis	LEMO EGG.00.302.CLL	1	LEMO FFS.00.250.CTA	1
HV GND, X axis		5		2
HV, actuator, Y axis	LEMO EGG.00.302.CLL	2	LEMO FFS.00.250.CTA	1
HV GND, Y axis		5		2
HV, common reference	LEMO EGG.00.302.CLL	3	LEMO FFS.00.250.CTA	1
HV GND, common reference		5		2
Strain gauge supply, X axis	LEMO EGG.00.304.CLL	1	LEMO FFA.0S.304 male	1
Strain gauge GND, X axis		4		4
Strain gauge +, X axis		3		3
Strain gauge -, X axis		2		2
Strain gauge supply, Y axis	LEMO EGG.00.304.CL	1	LEMO FFA.0S.304 male	1
Strain gauge GND, Y axis		4		4
Strain gauge +, Y axis		3		3
Strain gauge -, Y axis		2		2

Table 21 – UTT mirror interface. The strain gage connection on the HVC board uses a different connector due to space constrains. To connect the actuator, we will use a dedicated cable adaptor with 1-to-1 connection to '00' and '0S' connectors, or, preferably, the connector on the actuator end will be replaced with a '00' series.

Interface description	**From**	**To**	**Signal level**	**Connector**	**Pinout**
HVC input summing junction signals	Any external device (for maintenance and/or development purposes)	HVC board	-1.67V to 10V. The input signal is summed to the regular actuator command and amplified 12x. Input impedance >1MΩ.	SCSI-type 50 pin connector, female, on HVC analog and high voltage board	Pins 3,11,19,27,35,43: inputs channel Pins 1,9,17,25,33,41,49 and all even pins: GND
HVC diagnostic output signals	HVC board	Any external device (for maintenance and/or development purposes)	-2V to 10V. The output signal is 1/10 of the actuator output. Output dynamic impedance <10Ω.	SCSI-type 50 pin connector, female, on HVC analog and high voltage board	Pins 4,12,20,28,36,46: inputs channel Pins 1,9,17,25,33,41,49 and all even pins: GND
Strain gauges diagnostic output signals	HVC board	Any external device (for maintenance and/or development purposes)	Gain: 50mV/µm. Output dynamic impedance <10Ω.	SCSI-type 50 pin connector, female, on HVC analog and high voltage board	Pins 5,13,21,29,37,45: inputs channel Pins 1,9,17,25,33,41,49 and all even pins: GND

Table 22 – HVC board diagnostic interface. This interface is not used during standard operation, therefore it is routed only to an internal connector on HVC analog board. It is possible however to bring the signals externally by means of a ribbon connector.

6.4 Reliability

This chapter reports the preliminary reliability analysis performed on NGWFC RTC. The reliability is based on prediction according to standard methods for what concerns the proprietary electronics. For COTS components, the data reported by the manufacturer have been used.

6.4.1 MGAOS system reliability prediction

This chapter reports the preliminary reliability analysis performed on MGAOS electronics.

The reliability prediction has been computed according to the recommendation of [RD3], Notice 2, Parts Count Method. This method has been used to predict the reliability of each board, and then the partial result have been combined to compute the reliability of the complete control system.

The following conditions have been assumed:

- Operating environment: *ground, benign*. MGAOS will operate in a well protected environment, at moderate temperature and low humidity. There is no significant mechanical stress on the system.
- In most of the cases, the actual reliability factor published by the manufacturer of each electronic component has been used. The values reported for 55°C temperature and 90% confidence have been used. When actual production data were not available (e.g. resistors, connectors), the computation has been based on typical values recommended by [RD3] for the component type.
- The reliability data of the selected high voltage OP-AMP (Apex PA243) are not available. Data referring to amplifier from the same manufacturer with comparable electrical characteristics range from FITs ≅ 400 for MIL-STD screened versions to FITs ≅ 4500 for industrial versions. However one should take into consideration that the parts of which reliability data are available are all hybrids, while the PA243 device is monolithic. It is reasonable to expect a better reliability on the monolithic part, therefore a value of FITs=500 has been preliminarily assumed for this component.

Qty	Part Number	FITs, single part	FITs, total	Contribution
3	AdOpt DSP board	981	2943	33.3%
1	AdOpt BCU Board	503	503	5.7%
1	AdOpt Backplane Board	154	154	1.7%
1	AdOpt HVC board	4895	4895	55.4%
1	WFS AIA to BCU interface board	147	147	1.7%
1	BCU to DM HVA interface board	147	147	1.7%
160	Flat cables single connections	0.26	42	0.5%
		FITs	8830.6	

Computed MTBF (hours)	113,243

Table 23 – MGAOS electronics reliability prediction

6.4.2 Complete VME crate reliability prediction

The partial results obtained for the MGAOS system has been combined with the data of the COTS VME components using the same approach.

The assumptions have been taken:

- Reliability data for the Symmetricom IRIG timing board are not available. The same reliability of the CPU board has been assumed.
- STRAP reliability has not been taken into account, considering that this device is a pre-existing component, not being strictly part of the RTC system. In this perspective, it has been considered similarly to the WFS camera or to the DM HVA.

Qty	Part Number	FITs, single part	FITs, total	Contribution
1	MGAOS	8,831	8,831	24.6%
1	VME CPU board	5,605	5,605	15.6%
1	Symmetricom IRIG board	5,605	5,605	15.6%
1	VME/MGAOS PSU	3,584	3,584	10.0%
1	HV PSU	12,325	12,325	34.3%
		FITs	35,950	

Computed MTBF (hours)	27,816

Table 24 – Complete RTC crate reliability prediction

6.4.3 TRS system reliability

The declared MTBF of the Triton disk array is 300,000 hours, under normal operating conditions (ground, benign). There are no data available for the TRS server.

The actual disk array reliability might be an issue due to the reduced atmospheric pressure at site altitude. This might reduce significantly the lifetime of the disks heads. The disks chosen for this application have fluid-dynamic bearings, instead of air ones. This should guarantee an increased reliability under low pressure operating conditions. Anyway, this is a critical aspect requiring further investigation and, possibly, experimental tests.

6.5 Hardware maintenance

6.5.1 Periodic maintenance

6.5.1.1 VME and MGAOS crates

Within the system crate, the only parts subjected to mechanical stress are the cooling fans. Therefore it is recommended to check periodically (indicatively every 6 months) their functionality.

It is also recommended to check the PSU output voltages trimming every year.

The other components do not require scheduled maintenance.

6.5.1.2 MGAOS boards replacement

All MGAOS DSP and HVC boards can be replaced without requiring any preliminary configuration or calibration. The correct board address is automatically detected and configured at system startup on base of the position occupied by the board on the crate.

The HVC board analog calibration parameters are permanently stored on the local Flash memory, and loaded at system startup.

The only configuration required for MGAOS board replacement is the IP address setting of the BCU board. This can be easily done by sending a dedicated Ethernet command to the VME CPU board.

6.5.2 Spares

The RTC electronics spare list has been computed considering a lifetime of 10 years, 12 hours/day operating time, and on base of the reliability data reported in Table 23 and Table 24. We have considered two system operating simultaneously. The number of spares has been estimated imposing a 90% confidence level. The reported numbers do not take into account the availability of the third spare system, which is part of the deliverables.

Qty	Part Number	MTBF (single board)	Spare parts, 90% confidence (total for 2 systems)
6	AdOpt DSP board	1,019,368	1
2	AdOpt BCU Board	1,988,072	1
2	AdOpt Backplane Board	6,493,506	0
2	AdOpt HVC board	204,290	1
2	WFS AIA to BCU interface board	6,802,721	1
2	BCU to DM HVA interface board	6,802,721	1
2	VME CPU board	178,412	1
2	Symmetricom IRIG board	178,412	1
2	VME/MGAOS PSU	279,018	1
2	HV PSU	81,136	2

Table 25 – List of spares for 10 years of operation of two systems, 90% confidence.

All spares are considered as ‘line replaceable units’, i.e. all components can be simply replaced in short time (typ < 30 min), without requiring any particular operation to be performed.

7 SOFTWARE DESIGN

A very clear overview of the software blocks and related data flow is given in [AD4], §5.2. From the implementation point of view can distinguish three different software branches.

- MGAOS resident software, where all strictly real-time computations are performed (WFP, DTT/UTT controller)
- MVME6100 resident software, implementing the interface between WCP and RTC and housekeeping functions (WIF). The WCP is also implemented on the same CPU, but is under Keck responsibility
- TRS software, managing the telemetry database storage and query (TRS)

In the following sections we analyze implementation and performances of the above mentioned main software categories.

7.1 MGAOS Software

The MGAOS resident software is responsible of almost all WFP computation. The tasks computed by the DSP software are the following:

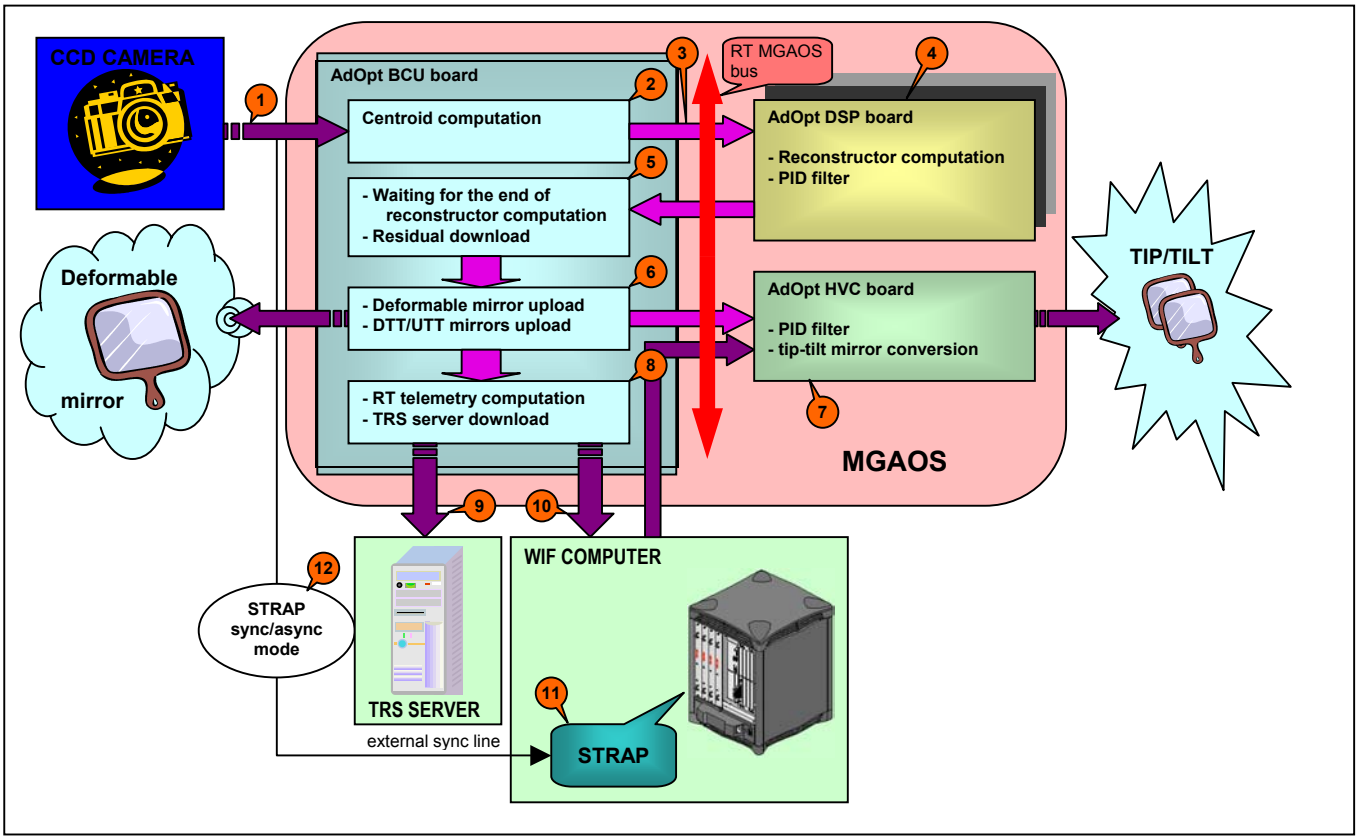
- Centroids computation
- Residual Wavefront computation
- Control law servo computation
- Real-time telemetry computation
- HVC real time control software

In the next paragraphs each task will be separately analyzed. Afterwards in Table 39 the interaction between all the various WFP computational parts will be described.

7.1.1 MGAOS data flow

Before entering in the reconstructor algorithm details, it is important to describe the data flux and the interaction between the various RTC components.

The following figure describes the various parts of the MGAOS and the interaction with the external devices:



Description of the flow chart points:

1) The raw pixels are downloaded to the MGAOS system using the standard AIA protocol interface. The SciMeasure CCD controller is able to acquire and transfer the pixels frame up to 2000 Hz.
   A dedicated interface is implemented into the AdOpt BCU board in order to put the raw pixels in the right order inside the DSP memory, skip the unused pixels and perform the centroid computation in parallel during the pixel download. The transfer time depends on the CCD configuration (frame rate and binning) but generally needs the 80% of the frame rate period.
2) The centroid algorithm, as mentioned in 7.1.2, includes raw pixel correction, computation of the two centroids for each sub-aperture and final centroids correction. The worst case duration is about 540 µs compared to a CCD integration time of 1 ms; therefore, thanks to the pipelined computation, the time latency introduced by this part is close to zero.
3) Once the centroids are all computed they are sent to the AdOpt DSP boards to proceed with the residual wavefront computation. The centroids vector is transferred through the real time MGAOS bus, which is able to transfer data up to 4Gbit/s. In this case the entire vector is sent in less than 5 µs.
4) The reconstructor algorithm adopted, as indicated in 7.1.3, includes the matrix product of the centroids vector by the reconstructor matrix to derive the residual wavefront vector. For each vector element a digital 3rd order PID is implemented, see 7.1.4. All this part of the algorithm is parallelized using the 6 DSPs available on the 3 AdOpt DSP boards and the estimated processing time is very short (65 µs). For this reason we decided to execute it after the centroids computation, rather than in pipelined mode.
5) While the AdOpt DSP boards are computing the reconstructor algorithm, the AdOpt BCU board waits for the end of the computation, polling on a flag on the AdOpt DSP board. When the new wavefront vector is ready it is read by the AdOpt BCU. Data are transferred using the real time MGAOS bus; the transfer takes about 4 µs.

6) Now the first 349 elements of the residual wavefront vector are remapped to the corresponding mirror channels and sent to the deformable mirror. A dedicated 16 bits parallel interface, implemented in the BCU FPGA, allows to send the data to the Xinetics high voltage drive. The interface is specified up to 10MHz, therefore the transfer time for all 384 channels amounts to 38.4µs.
   Depending on the system mode used, the elements #349 and #350 of the residual wavefront vector are sent to the AdOpt HVC board to update either DTT or UTT mirror position. These data are sent using the real time MGAOS bus requiring a negligible time.
7) The AdOpt HVC board is a dedicated to the control of DTT and UTT. By accessing the board through the MGAOS backplane, it is possible to update the desired tip-tilt mirror position. The tip-tilt reference is pre-processed to the algorithms indicated in 7.1.5.1 and 7.1.5.2 and the new position is passed to the high voltage drives.
8) When the time-critical part of the control loop is finished, the AdOpt BCU board can update the averaged telemetry data and prepare the full data stream to be sent to the TRS server.
9) The MGAOS is connected to the TRS through an optical FibreChannel link with a theoretical throughput of 1.7Gbit/s. The actual throughput, including packet formatting and software overhead, has been estimated to be ~50MB/s, with a significant contingency with respect to the actual worst-case data transfer requirement of 13MB/s.
10) The MGAOS is connected to the WIF through a 1Gbit/s Ethernet link. This line is used to read the averaged telemetry data when it is ready. The required data throughput is definitely negligible when compared to the available bandwidth.
11) When the RTC is used either in LGS or dual-NGS mode (see 7.1.5), the STRAP device, installed in the VME crate, is used to acquire the APD sensors. When a new set of counts is ready, STRAP notifies the WIF, which reads the new data and sends them directly to the AdOpt HVC board. The HVC board computes the APD centroids and executes a pre-processing algorithm, as indicated in 7.1.5.1; finally, the new DTT mirror position is passed to the high voltage drive.
12) STRAP can operate in two modes, synchronous and asynchronous. In the first case, STRAP sensor acquisition and centroids computation is triggered through the external ‘start of frame’ generated by the CCD camera. In the second case, STRAP works with a completely independent, internally generated rate.

7.1.2 Centroids computation

Centroids computation are performed by the DSP of AdOpt BCU board. The pixels are read sequentially from the CCD and stored in local DSP RAM following a user defined look up table. This look up table depends on the selected wavefront operating mode. The goal of this look-up table is to group pixels which form one sub-aperture. Pixels are ordered to achieve high DSP performance in the next computation steps.

To explain the principle of operation, a possible organization of the user-definable LUT is given in Table 26.

The BCU FPGA reads the pixels from the SciMeasure Little Joe. While the pixels are acquired, they are automatically transferred to BCU DSP memory by means of a DMA process and stored into the appropriate memory location, optimized for guaranteeing the maximum DSP computational throughput. Every 8 pixels, the number of centroids that can be computed with the already acquired pixels is transferred to a fixed DSP memory location.

Gain and offset compensation and centroid computation are executed by a background routine. The number of *computable centroids* is checked vs. the number of already computed centroids, and, if the former is greater than the latter, the DSP computes a set of centroids and returns to the previous check.

This implementation allows obtaining a very efficient code execution on the DSP. Setting properly the ‘number of computable centroids’ in the LUT one can control pipelined operation. Moreover, by simply modifying the LUT, the system can be easily adapted to different wavefront sensor configurations.

Memory address of 1st acquired pixel
Memory address of 2nd acquired pixel
Memory address of 3rd acquired pixel
Memory address of 4th acquired pixel
Memory address of 5th acquired pixel
Memory address of 6th acquired pixel
Memory address of 7th acquired pixel
Memory address of 8th acquired pixel
Number of *computable centroids* at this point
Memory address of 9th acquired pixel
Memory address of 10th acquired pixel
Memory address of 11th acquired pixel
Memory address of 12th acquired pixel
Memory address of 13th acquired pixel
Memory address of 14th acquired pixel
Memory address of 15th acquired pixel
Memory address of 16th acquired pixel
Number of *computable centroids* at this point
…

Table 26 – Pixel LUT content

The centroids are transferred to the DSP boards for residual wavefront computation. Data transfer is sequenced by the BCU board and the data are transmitted through the real-time bus by means of DMA processes without DSP software overhead.

It is possible to select between normal operation with sub-aperture flux normalization of centroids and denominator-free centroiding.

7.1.3 Residual Wavefront computation

This is the dominant computation in the entire WFP. Residual wavefront is computed as:

$$\{W\}=[R]\{C\}$$

R is a 608 x 352 matrix. The upper operation takes about 214 floating point kMACs. To achieve better performance and speed, this work is parallelized among the three DSP boards for a total amount of 6 DSPs. Each one is responsible of the computation of a sixth of the W vector.

In order to maximize the computational throughput of the DSP core and to exploit its hardware architecture, the total number of MAC operations performed by each DSP should be a multiple of four. In our configuration we have 6 DSPs, since the requested vector size is 352, this number is rounded up to 360 elements, so that each DSP performs a 608 x 60 matrix multiplication.

The R matrix is divided in 6 sub matrices: R_1, R_2, R_3, R_4, R_5, R_6.

$\{R_1\}$ takes rows 1 to 60 of $\{R\}$,

$\{R_2\}$ takes rows 61 to 120 of $\{R\}$,

$\{R_3\}$ takes rows 121 to 180 of $\{R\}$,

$\{R_4\}$ takes rows 181 to 240 of $\{R\}$,

$\{R_5\}$ takes rows 241 to 300 of {R},

$\{R_6\}$ takes rows 301 to 352 of {R}, and the remaining rows are zeroed.

The 6 resulting operations are:

$\{W_1\}=[R_1]\{C\}$ assigned to DSP #1

$\{W_2\}=[R_2]\{C\}$ assigned to DSP #2

$\{W_3\}=[R_3]\{C\}$ assigned to DSP #3

$\{W_4\}=[R_5]\{C\}$ assigned to DSP #4

$\{W_5\}=[R_5]\{C\}$ assigned to DSP #5

$\{W_6\}=[R_6]\{C\}$ assigned to DSP #6

Each W_i is a 60 element vector, together they form the full 352 element W vector. From W_6 vector the last eight elements are simply ignored.

7.1.4 Control law servo computation

Like the residual wavefront also the control law computation is split among DSP boards 1, 2 and 3.

Each DSP is responsible of one sixth of the full command vector computation which is computed as:

$$C_i[n] = - b_1C_i[n-1] - b_2C_i[n-2] - b_3C_i[n-3] + a_0W_i[n] + a_1W_i[n-1] + a_2W_i[n-2] + a_3W_i[n-3]$$

where the input *W* is the residual wavefront vector and the output *C* is the output vector for the deformable mirror. The index *i* refers to each DSP (1 to 6). Also in this case the last eight elements of C_6 sub-vector are simply ignored. The *a* and *b* values are the control law coefficients. The indexes *n, n-1, n-2* and *n-3* refer the current value and the three previous values of each vector.

Actuator commands are clamped to a maximum admissible value, and the occurrence of threshold exceeding is recorded by incrementing a dedicated counter. If the control loop is closed the computed command is the new position that will be sent to the mirror otherwise, if the control loop is open, only the DM origin vector (nominal flat position) will be sent to the mirror.

Once the output commands for the DM are available, they are normalized, converted to the appropriate unsigned 16-bit DAC values and transferred to the BCU in a single read operation through the real-time backplane. Finally, from the BCU PIO port, the DM commands are transmitted to the Xinetics HVA. The BCU FPGA implements the real-time FPDP-like interface to the HVA. The RMS wavefront error is also computed by the DSP of BCU.

7.1.5 HVC real-time control software

The HVC board controls the DTT/UTT mirror outputs.

The use of this board depends on system setup according to the following table:

Mode	DTT Mirror	UTT Mirror
NGS	Used, driven by WFP	Not used
LGS	Used, driven by STRAP (synchronous with frame rate)	Used, driven by WFP
Dual NGS	Used, driven by STRAP (not synchronous with frame rate)	Not used

Table 27 -Tip-tilt operating modes. The computation step refer to the algorithm description in §7.1.5.1.

The HVC software can be divided into two main tasks:

- tip-tilt mirror correction before applying the command. This task waits for a tip-tilt mirror command update, when it arrives it performs the two different pre-correction algorithms for the DTT and UTT mirrors as described later and sends the results to the local mirror servo control loop.
- local mirror servo control loop. This interrupt control routine is an high speed digital servo loop (running at ~70kHz) which controls the HV outputs in order to obtain the desired tip-tilt mirror movement requested from the first task. According to the high performances required by this routine, it has been decided to implement it using assembler code.

7.1.5.1 DTT mirror correction

The DTT mirror command source is indicated in Table 27. The first four steps of the algorithm are executed only if the input data arrives from STRAP, which sends the four APD counts. In NGS mode data arrive from the WFP (elements #349 and #350 of the residual wavefront vector); in this case the algorithm begins directly from the step #5.

Hereafter the sequence of computational steps:

Step #1: sky background correction:

- $\tilde{A}_i = A_i - S_i$ where $i = 1,2,3,4$

Step #2: total flux computation and threshold check:

- $tot\_flux = \sum \tilde{A}_i$ - if $tot\_flux < flux\_threshold$ then $tot\_flux = flux\_threshold$

Step #3: APD centroids computation:

- $C_{Ax} = \dfrac{A_1 - A_2 - A_3 + A_4}{tot\_flux}$
- $C_{Ay} = \dfrac{-A_1 - A_2 + A_3 + A_4}{tot\_flux}$

Step #4: APD centroids interaction matrix correction:

- $\begin{bmatrix} C_{ix} \\ C_{iy} \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{2,1} \\ m_{1,2} & m_{2,2} \end{bmatrix} \begin{bmatrix} C_{Ax} \\ C_{Ay} \end{bmatrix}$

Step #5: angular offset correction:

- $C_x = C_{ix} - C_{ox}$
- $C_y = C_{iy} - C_{oy}$

Step #6: control law servo computation in the discrete domain (3/3 order) applied to each centroid element:

- $m[n] = -b_1 m[n-1] - b_2 m[n-2] - b_3 m[n-3] + a_0 c[n] + a_1 c[n-1] + a_2 c[n-2] + a_3 c[n-3].$
  where: c is the generic input centroid element and m is the generic output element of the filter, which is the raw mirror output command;

Step #7: mirror output command offsetting:

- $\widetilde{M} = M + M_{off}$

Step #7: disturbance history optional correction:

- $M_f = \widetilde{M} + M_d[i]$

Step #8: DTT mirror 3 axis splitting:

- $$\begin{bmatrix} M_0 \\ M_1 \\ M_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\sin(\frac{\pi}{3}) & -\sin(\frac{\pi}{6}) \\ \sin(\frac{\pi}{3}) & -\sin(\frac{\pi}{6}) \end{bmatrix} \begin{bmatrix} M_{fx} \\ M_{fy} \end{bmatrix}$$

After the correction is computed the new three actuator mirror commands are passed to local actuator servo control loop to perform the desired tip-tilt mirror movement.

7.1.5.2 UTT mirror correction

The UTT mirror is used only in LGS mode.

The tip-tilt inputs are computed by the WFP (elements #349 and #350 of the residual wavefront vector).

Hereafter the sequence of computational steps:

Step #1: angular offset correction:

- $C_x = C_{ix} - C_{ox}$
- $C_y = C_{iy} - C_{oy}$

Step #2: control law servo computation in the discrete domain (3/3 order) applied to each centroid element:

- $m[n] = -b_1 m[n-1] - b_2 m[n-2] - b_3 m[n-3] + a_0 c[n] + a_1 c[n-1] + a_2 c[n-2] + a_3 c[n-3]$.
  where: c is the generic input centroid element and m is the generic output element of the filter, which is the mirror output command;

Step #3: mirror output command offsetting:

- $\widetilde{M} = M + M_{off}$

Step #4: disturbance history optional correction:

- $M_f = \widetilde{M} + M_d[i]$

Step #5: UTT axis rotation:

- $$\begin{bmatrix} M_0 \\ M_1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} M_{fx} \\ M_{fy} \end{bmatrix}$$

After computing the correction, the new actuator mirror commands are passed to the local actuator servo control loop to perform the desired tip-tilt mirror movement.

7.1.5.3 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 filter with 3 taps IIR filter on the position error and 3 taps IIR filter 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 DTT mirror uses three channels, while the UTT mirror makes use of two channels and a third channel is driven at constant voltage, which definable through WIF. During operation this voltage is typically set at mid output range.

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.

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

Tuning of servo-loop is possible by means of automated tuning scripts, based on Matlab routines. A direct communication interface between Matlab and MGAOS is available. Matlab scripts command the HVC board to perform dynamic response tests. To do this, time-sequences can be uploaded to and downloaded from the HVC board. The sequences are executed and acquired at the same speed of the local servo-loop (~70kHz). The tuning algorithm is based on function-cost minimization algorithms, as explained in 6.1.1.3.4.

7.1.5.4 Automatic power up and shutdown procedures

In order to avoid excessive mechanical stress to the mirrors and/or undefined output states during system power up, a 'smooth' power up and shutdown sequence is foreseen.

The power-up procedure is described hereafter:

- all HV relays open
- switch on HV supply
- command all HV outputs to 0V
- close (now safely) HV output relays
- check actual outputs with embedded diagnostics
- rise smoothly (under SW control) the voltages. In particular, the UTT mirror will be driven so that the common voltage is always kept at 2x command voltages while the high voltage is rising. This will bring smoothly the mirror to mid range position.
- close control loop on local position
- ready

The shutdown procedure is described hereafter:

- bring mirror to mid range position under closed loop
- open loop and apply same voltages in open loop

- decrease smoothly (under SW control) the voltages. In particular, the UTT mirror will be driven so that the common voltage is always kept at 2x command voltages while the high voltage is going to zero. This will bring smoothly the mirror to off position
- check actual outputs with embedded diagnostics (shall be all zero)
- open HV relays
- switch off HV supply

7.1.6 Telemetry data

The scope of the telemetry data is to monitor the WFC system performance and status, to allow offloading of large adaptive optics corrections to the telescope control system and to reconstruct the system history by means of post processing.

For this reason two different kinds of telemetry data are available on the NGWFC, called averaged telemetry and full data telemetry:

- The averaged telemetry computes the average of the most important data with a user defined rate (typically 1 second). When available, the WCP reads these data to monitor the WFC system during operation.
- The full data telemetry saves most of the real-time data stream to a big data repository (TRS). The complete system configuration and all applied configuration changes are also saved by TRS. Therefore, it is always possible to reconstruct the past system functioning. The TRS can save about 50 hours of system operation at maximum frame rate.

7.1.6.1 Averaged telemetry computation

The averaged telemetry data can be distinguished in two groups: *continuous telemetry* data stream and *on demand* telemetry data stream. This telemetry data can be retrieved through the WIF interface (§7.2.2).

We report hereafter a list of all the telemetry streams computed online.

Continuous data stream is computed accumulating samples until a user definable period has expired, then the average is calculated and the result saved to be read by WIF. Automatically a new data accumulation begins until the following period expires and so on. Some continuous data streams are performed by the DSP of the AdOpt BCU board and others by MVME CPU board.

The following continuous time averaged values are computed by the AdOpt BCU board:

- *Tip-Tilt mirror command* (DTT or UTT according to the tip-tilt operating mode adopted) derived from the high voltage output command of the AdOpt HVC board. *DM Zernike focus* derived from the DM position vector.
- *WFS centroid vector* derived from the first 349 elements of the residual wavefront vector.
- *WFS Zernike tip-tilt* derived from the elements #349 and #350 of the residual wavefront vector.
- *WFS Zernike focus* derived from the element #351 of the residual wavefront vector.
- *WFS subaperture intensity values* derived from the 304 subaperture intensity values.

The following continuous time averaged values are computed by a dedicated real-time task on MVME CPU board. They are used when the DTT mirror command is calculated through the STRAP sensor device:

- *DTT mirror command* derived from the high voltage output command of the AdOpt HVC board.
- *APDs sensor centroids* derived from the acquired four APD counts.

On demand data stream is computed similarly to continuous data stream but the accumulation starts by means of a WIF command. A single accumulation and average computation is performed until a new request arrives.

This telemetry is computed by the AdOpt BCU board, the following output values are available on demand:

- *CCD raw frames*

7.1.6.2 Full data telemetry management

The full data telemetry is stored to TRS for off-line analysis and data retrieval. The telemetry data have been divided in four streams, according to the different update rates. This subdivision corresponds also to four different databases allocated on the TRS (see §7.3.2). The four streams are reported on Table 28:

Telemetry stream name	**Source**	**Rate**	**Transfer over:**
Full Frame rate telemetry Buffer (FFB)	MGAOS	Full frame rate	FibreChannel from MGAOS to TRS
Decimated Frame rate telemetry Buffer (DFB)	MGAOS	Decimated frame rate (up to 100 Hz)	FibreChannel from MGAOS to TRS
STRAP diagnostic	STRAP	Full STRAP acquisition rate (may be synchronous or not to frame rate, see Table 27)	VME backplane to VME CPU, then Ethernet from VME CPU to TRS
Configuration	VME CPU	On change	Ethernet from VME CPU to TRS

Table 28 - Telemetry buffer types

The details of FFB and DFB are reported in this section, while *STRAP diagnostic* and *Configuration* streams are reported in §**Errore. L'origine riferimento non è stata trovata.** and §7.2.5 respectively.

FFB and DFB data are first buffered and prepared in the WFP, taking into account that TRS is not a real time machine, and also in order to optimize communication between WFP and TRS.

In Table 29 all fields of FFB record are listed:

Section	**Field Name**	**Type**	**Size (byte)**
Header	Telemetry Buffer Type	Uint16	2
	Record Length	Uint16	2
	Timestamp	Uint64	8
	Frame Counter	Uint32	4
Data	Subaperture intensity vector	Uint32	304 x 4
	Offset centroid vector	Float32	608 x 4
	Residual wavefront error vector	Float32	349 x 4
	Residual TT & Focus	Float32	3 x 4
	Residual RMS	Float32	1 x 4
	DM actuator vector	Float32	349 x 4
	Tip-Tilt mirror command data (DTT or UTT)	Float32	3 x 4
	CLMP strain gauge (DTT or UTT)	Float32	3 x 4

	Conf-id	Uint32	4
Footer	Telemetry Buffer Type	Uint16	2
	Record Length	Uint16	2
	Timestamp	Uint64	8
	Frame Counter	Uint32	4
Total size			6516
Throughput @ 2Khz (CCD-39)			12.4 MB/s
Throughput @ 1Khz (CCiD-56)			6.2 MB/s

Table 29 - FFB record description

The Dummy field is an implementation detail which allows a 16 byte alignment of the various records. At each real-time computational cycle, all data for telemetry are transferred from the DSP memory of all AdOpt DSP board to the BCU and here these data are stored into a transfer queue located on SDRAM.

The DFB is running at reduced frame rate (up to 100 Hz) and retrieves the Raw CCD pixels from BCU DSP Ram. Also this transfer queue resides in BCU SDRAM. The single records of this buffer are listed in Table 30:

Section	**Field Name**	**Type**	**Size (byte) CCD-39**	**Size (byte) CCiD-59**
Header	Telemetry Buffer Type	Uint32	4	4
	Timestamp	Uint64	8	8
	Record Length	Uint32	4	4
Data	Raw CCD data (from 1600 to 25600 pixels, after binning)	Uint32	Max. 6400 (80x80x16 bit)	Max. 51200 (160x160x16 bit)
Footer	Telemetry Buffer Type	Uint32	4	4
	Timestamp	Uint64	8	8
	Record Length	Uint32	4	4
Total size			6432	51232
Throughput @ 100Hz			0.6 MB/s	4.9 MB/s

Table 30 - DFB record description

At each new frame, the BCU sequences the transfer of data from its local SDRAM to the TRS by means of the FibreChannel link. As already mentioned above, a strict time determinism on the TRS interface is not required, simply it has to be guaranteed that the mean data rate of ~13 MB/s is achieved. This rate is the highest throughput needed and is given when the CCD-39 is used at 2kHz (0.6 MB/s + 12.4MB/s).

7.1.6.2.1 WFP–TRS data transfer over FibreChannel

The communication between WFP–TRS goes over FibreChannel using IP protocol.

This is a standard protocol supported by many FibreChannel devices, and is also supported by the Qlogic HBA device installed on the TRS. An overview of the implementation details is given in Table 31.

The basic idea of IP over FibreChannel is to encapsulate standard IP packets into the data field of FibreChannel frames (see also [RD6]).

Standard	Name	Size (bytes)	Description
FibreChannel	SOF	4	Start of FC Frame
FibreChannel	Frame Header	24	FC Frame Header
FibreChannel	Network Header	16	Network header (is part of FC Frame payload)
FibreChannel	LCC/SNAP Header	8	LCC/SNAP Header
IP	IP Header	20	IP Header
MG_DTP	Packet Index	2	Packet main index to distinguish each single diagnostic frame
MG_DTP	Packet Sub Index	2	Single packet index to allow merging all single IP packets belonging to a single diagnostic frame
MG_DTP	Packet Size	2	Size of single MGDF packet
MG_DTP	Packet Total Size	2	Total size of current diagnostic frame
MG_DF	Header Telemetry Buffer Type	2	Buffer Type to distinguish diagnostic buffer records (DFB, FFB, etc)
MG_DF	Header Timestamp	8	See §7.1.8
MG_DF	Header Record Length	4	
MG_DF	Header Dummy	2	To align to 16 bytes
MG_DF	Data…	…	
MG_DF	Footer Telemetry Buffer Type	2	All Footer data equals Header data this is for robustness purposes
MG_DF	Footer Timestamp	8	
MG_DF	Footer Record Length	4	
MG_DF	Footer Dummy	2	
FibreChannel	CRC	4	Checksum
FibreChannel	EOF	4	End of Frame

Table 31 - FC-IP encapsulation structure for FFB and DFB streams

The data transmission overhead of this protocol is about 6% for the FFB record. The overhead has been computed taking into account all protocol specific fields and all dummy bytes needed for alignment purposes.

The FibreChannel interface to TRS represents an important aspect of the development and implementation process, in fact the FibreChannel interface already available on AdOpt systems does not support the FC2 and higher layers. These (up to layer FC4) are necessary for IP over FibreChannel. In order to prove the feasibility of this concept, we have implemented a point to point connection between BCU board and a standard PCI FibreChannel host adapter interface (QLogic type 2312 Host interface board, same type as the one foreseen for final communication between TRS and BCU and TRS to disk array). For testing purposes, we developed the interface up to the ICMP packet level (*ping*). By sending sequences of large *ping* packets, we obtained a throughput of ~50MB/s, with a significant contingency with respect to the actual worst-case data transfer requirement of 13MB/s.

Both test logic and software have been developed with the goal of minimizing the effort to expand them to the final FC over IP implementation.

7.1.6.2.2 MVME 6100 – TRS data transfer

The communication between MVME 6100 and TRS goes over standard TCP/IP protocol using the standard LAN. The data transfer rates on this communication link are quite small. We can distinguish three streams:

- RTC configuration
- DTT/STRAP data
- non-RTC telemetry

RTC configuration and STRAP data will be transferred using a single TCP/IP socket connecting WIF and TRS. Table 32 shows the base data packet of the WIF-TRS link.

Non-RTC telemetry is a record capability provided by the TRS for configuration data referring to devices outside of the NGWFC such as OBS, SC, NIRC2. This data are sent to the TRS by the WCP. A dedicated socket is used for this purpose. The data transfer rate for this stream is small: about 1kB/s (256 bytes at 5Hz). Table 33 shows the base data packet of the WCP-TRS link.

	MG_DF	Header Telemetry Buffer Type	uint16	Buffer Type to distinguish diagnostic buffer records (STRAP, CONFIG)
	MG_DF	Header Record Length	uint32	
	MG_DF	Data…	…	

Table 32 - TCP-IP data structure for STRAP and Configuration streams (payload only shown here)

	MG_DF	Header timestamp	uint64	
	MG_DF	Data…	char array	null-terminated ASCII string up to 256 bytes

Table 33 - TCP-IP data structure for non-RTC telemetry (payload only shown here)

7.1.7 On-the-fly parameter swapping

To allow on-the-fly parameter swapping the WFP reserves for each parameter two memory locations. Doing so the WFP has two parameter sets or areas: set A and set B. Switching from one to the other set occurs instantaneously at the beginning of each control cycle, if requested.

Only one parameter area at a time is used. The one currently in use is called *working area*, the other one is the *setup area*. All parameter configuration and control is performed by the CIE (Command Interpreter Executer) running on the MVME6100 (see also §7.2).

In order to know exactly with which configuration set WFP is operating, and to assure atomic parameter change, the parameter change sequence illustrated in Figure 29 has been adopted:

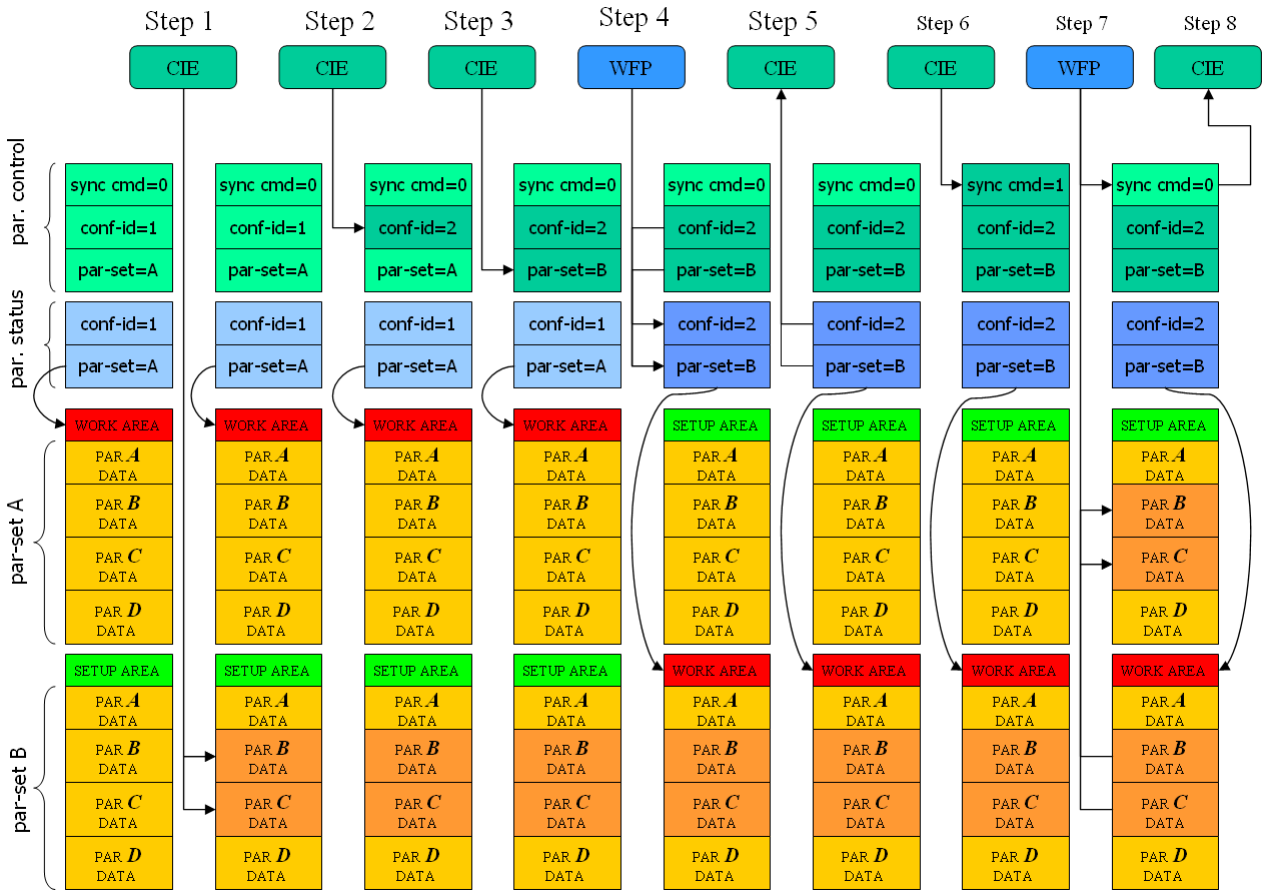


Figure 29 - WFP parameter on-fly swapping

Initially, both parameter areas, A and B, contain identical values. For simplicity we assume that WFP has only 4 parameter types parA, parB, parC, parD. Note that these parameters must not be scalars but can also be vectors or matrices, for instance pixel background, wavefront reconstructor matrix and so on.

To implement the sequence, two parameter areas are needed: parameter *control* area, and parameter *status* area.

The parameter control area is formed by the following fields

- par-set this field selects which parameter area set the WFP should use
- conf-id this is needed to uniquely identify each configuration adopted by the WFP (see §7.2.5)
- sync-cmd field triggers the BCU to synchronize the configuration areas. (working area is copied to the setup one when this flag is asserted by the CIE)

The parameter status area contains only the current conf-id field and the current par-set field, which reflects the actual parameter status of the WFP.

A parameter change is accomplished in the following steps:

- The CIE writes new parameters in the setup area. In the example, the setup area is the area B.
- When CIE has finished to upload the new parameters, it writes the conf-id into the parameter control area.
- Now everything is ready and the CIE switches the par-set field in the parameter control area.
- The WFP detects that the par-set field has changed. At this point the conf-id and par-set fields in the parameter status area are updated, and the WFP will use the new configuration. In this way the parameter switch is atomic, and the exact time when a certain parameter-set is used is known.

- Meanwhile the CIE polls the parameter status area and waits until it has been updated by the WFP to reflect those in the parameter control area. When both the conf-id fields are identical, the CIE can be confident that the parameter transaction has terminated successfully, and WFP is working with the new downloaded parameter set.
- Now the parameter transaction has finished. However, it might be convenient to retain the two parameter areas always identical. Doing so, it is assured that only the parameters affected by the transaction do really change. To avoid a lot of communication traffic in the WIF-WFP interface (Gbit Ethernet), the parameter areas can be synchronized directly by the BCU. To do so, the CIE writes into sync-cmd field.
- As soon as the BCU detects the trigger it will copy the working area to the setup area. When the BCU has finished, it will clear the sync-cmd field to signal the completion of synchronization.
- The CIE polls the sync-cmd field waiting for BCU copy completion. Now the CIE can go on with a new parameter transaction.

The last 3 steps are optional, and it will be the CIE who controls if the synchronization will be delegated to the BCU or if the CIE will do this. As a general rule, CIE will delegate the synchronization work for big parameters, i.e. large vectors and matrices, indeed for scalar parameter it will be more efficient if the setup area is synchronized by the CIE itself.

There could be operating conditions where it might be convenient to preserve two different configurations, switching between them continuously. The proposed mechanism allows to deal also with this situation, without having to upload the new configuration at each switch.

7.1.8 Time-stamping

Time stamping of diagnostic data within the MGAOS is based on an internal timer. This timer is synchronized with the absolute time reference (Symmetricom VME timing decoder) by sending a synchronous hardware signal (generated by the BCU board), to the Symmetricom board (time on event feature). The timestamp of this signal is then read by the MVME6100 board and sent to the MGAOS through the Ethernet interface. The synchronization can occur quite rarely: e.g., 2s period would guarantee an accuracy of <100μs on the internal time-stamping (this is referenced to a 50ppm internal oscillator), and the only requirement for the communication is that it shall occur before the next synchronization event.

In the case that STRAP is running in asynchronous mode, time-stamping of STRAP diagnostic data is performed by the VME CPU. Upon acceptance of the STRAP real-time communication interrupt, the VME CPU reads of the current time from the Symmetricom board (time on demand feature).

The timestamp is a 64 bit unsigned integer representing the number of μsec passed, since 01.01.2000

7.1.9 WFP memory requirements

In this section we analyze the memory requirements for real time data processing. One crucial aspect in order to exploit the computational throughput of the DSP is to guarantee that the computations are always performed on DSP internal memory, to avoid core stalls and latencies. This allows performing one complete operation (fetch, operation and store) in one single DSP clock achieving high rate of throughput. In particular, considering the vector multiply operation, the TigerSharc architecture allows to actually perform two complete 32bit floating point MACs in a single internal clock cycle, without stalls. Feasibility of this is discussed in Table 34, Table 35 and Table 36, where we can see that all data can be stored in DSP internal RAM.

7.1.9.1 BCU FPGA memory

Item	Description of memory occupation	Memory requirement in bits
Pixels look-up table	160x160 pixels x 2 copies x 9/8 (for instant swapping TBC) 16 bits	Stored in FPGA Ram (921600 << 3 Mbits available)

Table 34 - BCU FPGA memory requirements

7.1.9.2 BCU DSP memory

Item	Description of memory occupation	Variable type	Memory requirement in bits
Pixel input	160 x 160 x 2 copies (for pipelining)	uint16	819,200
Pixel background correction (reduced by $\pi/4$)	19456 x 2 copies (for instant swapping)	uint16	622,592
Pixel gain correction(reduced by $\pi/4$)	19456 x 2 copies (for instant swapping)	float	1,245,184
Sub-aperture intensity vector	304 x 2 copies (for pipelining)	uint32	19,456
User-defined sub-aperture normalization vector	304 x 2 copies (for instant swapping)	float	19,456
Centroid Origin	608 x 2 copies (for instant swapping)	float	38,912
Centroid Gain	608 x 2 copies (for instant swapping)	float	38,912
Offset centroid vector	608 x 2 copies (for pipelining)	float	38,912
Residual wavefront vector	360 x 2 copies (for pipelining)	float	23,040
DM actuator output vector	384 (total number of DM channels)	uint16	6,144
DM actuator map	349 (number of used DM channels)	uint16	5,584
DM actuator flat position	349 (number of used DM channels)	uint16	5,584
DM Focus modal vector	349 x 2 copies (for instant swapping)	float	22,336
DTT mirror position command	2 x 2 copies (for pipelining)	float	128
UTT mirror position command	2 x 2 copies (for pipelining)	float	128
DM Zernike Focus averaging	1 x 2 copies (for pipelining)	float	64
WFS centroids averaging	608 x 2 copies (for pipelining)	float	38,912
WFS Zernike Focus averaging	1 x 2 copies (for pipelining)	float	64
WFS Zernike tip-tilt averaging	2 x 2 copies (for pipelining)	float	128
WFS subaperture intensity averaging	304 x 2 copies (for pipelining)	float	19,456
Raw pixels averaging	160x160	float	819,200
Total memory requirement (BCU DSP)		3,783,392 (<< 6 Mbits)	

Table 35 – BCU DSP memory requirements

7.1.9.3 DSP, 3 boards

Item	Description of memory occupation	Variable type	Memory requirement in bits
Reconstructor matrix	60 x 608 x 2 copies (for instant swapping)	float	2,334,720
Residual wavefront vector	60	float	1,920
Filtered residual wavefront vector	60	float	1,920
A_0, A_1, A_2, A_3 and B_1, B_2, B_3 The control law matrixes/vectors	7 x 2 copies (for instant swapping)	float	448
Delay line (past samples) of PID input and output vector	60 x 7 x 2 copies (for instant swapping)	float	26,880
Total memory requirement			2,365,888 (<< 6 Mbits available)

Table 36 – DSP Boards #1, #2 and #3 memory requirements

7.1.10 Performance estimate

The computational requirements vs. computational power availability for the relevant real time computational processes are analyzed in Table 37. A detailed study has been done in order to investigate the possibility of using C language instead of Assembler language. Another scope of this study was to consolidate that the chosen hardware architecture allows implementing the algorithm with a large contingency for further upgrades.

7.1.10.1 Computational time requirement analysis

The following table summarize the computational time for each part of the real time algorithm, as measured using a preliminary but complete code. At this level we performed a non-exhaustive code optimization, leaving some room for further improvement, if necessary.

Real-time computation	Required computational time using C language	Required computational time using assembler language	Notes
Pixel background, flat field, centroid computation (BCU)	540 µs	300 µs	This is the worst condition using the CCID56 computing 608 centroids with the sub-aperture size of 8x8
Matrix vector multiply and control law servo (DSP)	65 µs “time critical” followed by 10 µs not time critical	62 µs “time critical” followed by 8 µs not time critical	The C code is similar to the assembler code because the compiler optimizes the matrix product as in assembler

Table 37 – Required vs. available computational power

7.1.10.2 Data transfer requirement analysis

The data transfer requirements vs. actual bandwidth availability for the relevant real time and diagnostic data transfers are analyzed in Table 38. The contingency appears to be adequate in all considered data transfer processes.

Real-time data transfer	**Amount of data (each control cycle)**	**Expected throughput**	**Notes**
Raw pixel data transfer from AIA interface to DSP memory	160 x 160 x 16 = 409,600 bits	4 Gbits/s (64 bits parallel DSP cluster bus)	Transfer between AIA port and DSP is handled by the FPGA. Data transfer is pipelined with the pixel acquisition and the centroids computation, so the latency does not contribute to the processing time.
Centroids transfer from BCU to DSP boards #1, #2 and #3	608 x 32 = 19,456 bits	4 Gbit/s (real-time bus on proprietary backlane)	This operation is performed in a single shot using the broadcast primitive on the real-time bus.
DM commands transfer from DSPs to DM HVA	349 x 16 = 5,584 bits	The lowest between 4 Gbits/s (real-time bus on proprietary backplane) and 256 Mbits/s (DM HVA interface). Clearly DM HVA interface is the limiting factor.	Most of the time is spent by the HV DMA interface
Data transfer from STRAP to DTT	Negligible	VME interface and Gbit Ethernet	Not fully deterministic, we assume 120µs
Diagnostic data transfer from DSP boards to BCU board	360 x 32 (WF residuals) + 360 x 32 (DM command) = 23,040 bits	4 Gbit/s (real-time bus on proprietary backlane)	
Diagnostic data transfer from DSP to SDRAM of BCU board	FFB record = 52,640 bits DFB record = 409,856 bits	2 Gbit/s internal BCU diagnostic buffer	For the DFB record size we considered the worst case of 160x160 no binning CCD setup
Diagnostic data transfer from BCU board to TSR disk array	FFB record = 52,640 bits DFB record = 409,856 bits	2.125 Gbit/s (FibreChannel) -> 1.7Gbit/s (real throughput)	Respect to the theoretic throughput the 8/10 bit encoding/decoding and the pay load data respect the entire FC-IP packet size should be considered

Table 38 – Data transfer latencies expectation

7.1.10.3 Real time system timing diagram

The real time computation sequence and its timing are analyzed in Figure 30 and Table 39, including the interactions and data exchanges involved in the different computational steps. The sequence refers to a single CCD frame.

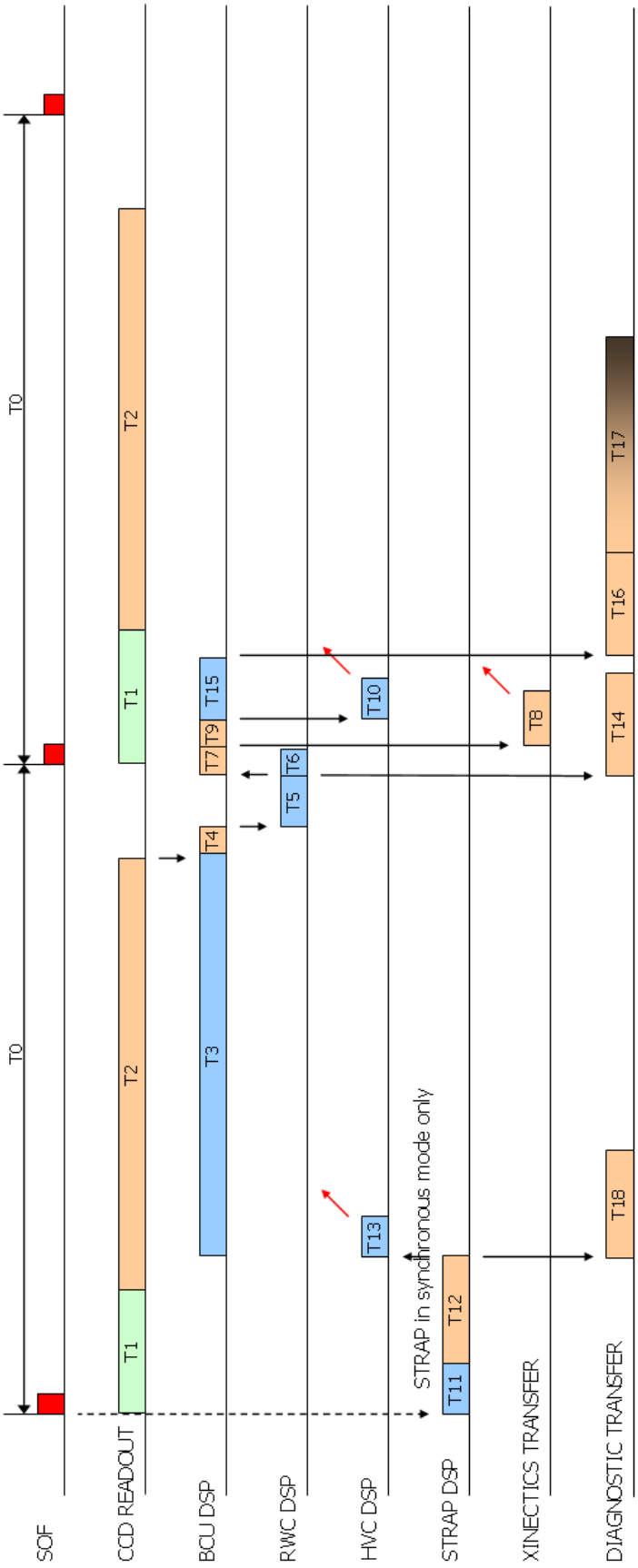


Figure 30 - Real time sequence

Step	Description	Requested time
T0	The sequence starts with the start of frame (SOF) sent by the CCD controller to the BCU	depends on the CCD controller setup, down to 500 μs
T1	Hold period before the CCD controller sends the first pixel	depends on the CCD controller setup (150 μs for 80x80 CCD @ 1KHz)
T2	Pixel read-out phase, during the pixel up-loading the BCU is able to redirect the pixels to the correct position to the DSPs memory through a preloaded look-up table in order to optimize the centroid computation in the DSP program.	depends to the CCD controller setup (700 μs for 80x80 CCD @ 1KHz)
T3	Centroid computation phase, in order to increase the computation performances the centroid computation, this operation is pipelined during pixel reading from CCD. As reported in 7.1.2 the centroids computational time is less than the CCD read out time.	the code has to wait the CCD pixel downloading
T4	Centroid transfer to the DSP boards.	6 μs
T5	Residual wavefront computation and last tap of control law servo computation (with 3DSP boards setup). As above mentioned this computation is not pipelined because the data flow complication versus the time delay saved is not justified	65 μs
T6	Control law servo pre-computation for the next step (with 3DSP boards setup)	10 μs
T7	BCU read back of DM commands and residual wavefront vector from DSP boards	10 μs
T8	BCU write of DM commands to Xinetics DM. This phase does not require software overhead, apart a short initial time to start the mirror commands upload. DSP can continue processing meanwhile.	40 μs
T9	BCU write of residual tip-tilt to HVC board (to DTT mirror in NGS mode, to UTT mirror in LGS mode, see Table 27)	1 μs
T10	DTT or UTT mirror commands computation (depending on mode): offset, servo loop, disturbance, mirror offset, rotation, distribution over 3 act. (DTT mirror only)	2 μs
T11	STRAP acquires the APD quad-cell sensor	5 μs
T12	STRAP data transfer: VME IRQ response, transfer over VME bus to VME CPU and over Ethernet from VME CPU to BCU and to HV board	120 μs
T13	APD centroid and DTT mirror commands computation and distribution over 3 act	2 μs
T14	Decimated Frame Buffer (DFB) storage to BCU SDRAM. CCD raw pixels considering the worst case 160x160 no binning CCD setup	205 μs
T15	Continuous time averaged telemetry streams computation	15 μs
T16	Full Frame Buffer (FFB) storage to BCU SDRAM	26 μs
T17	FibreChannel telemetry data transfer to TRS	
	FFB	32 μs
	DFB (considering the worst case 160x160 no binning CCD setup)	260 μs
T18	STRAP telemetry data transfer to TRS through Ethernet	30 μs

Table 39 - Timing and real time operations sequence

The actual computational latency can be determined considering the operations performed after acquisition of last pixel from WFS. The pixel compensation and centroid processing are pipelined operations. In this way, the residual processing time after last pixel readout can be minimized down to less than **90µs**. After this, data transfer to DM mirror starts immediately. The total time from start-of-frame is strongly influenced by the SciMeasure Little Joe pixel transfer time.

7.2 MVME6100 Software

The software running on the MVME6100 is based on the VxWorks operating system and accomplishes the following main tasks:

- System initialization
- WIF interface
- STRAP and Timing support task
- System monitoring
- System configuration storage to TRS

The WCP software resides also on the MVME6100 but is not described in this document.

7.2.1 System start-up

At system start up the MVME 6100 downloads the VxWorks operating system from a workstation host through Ethernet. When the VxWorks operating system is ready two start-up scripts are launched:

- the first will initialize the WIF and MGAOS software which are developed by Microgate
- the second will initialize all the WCP software which is developed by Keck

The first script will initialize:

- TRS communication
- IRIG timing board
- WFP: code is uploaded to the DSPs
- MVME to MGAOS communication
- Command interpreter and executer (CIE)
- WCP/WIF interface
- WCP watchdog
- WFP monitoring task

Then the second script will be launched and WCP will be initialized.

After initialization is complete the RTC will switch to standby state, and the WIF will accept commands from the WCP. When the WCP issues the Init command (see 9.3) all parameters will be loaded from files stored on the host WS.

The system configuration is recorded on a main file (*.ini*) that refers to other files where the parameters referring to s particular subsystems are stored. Hereafter we present an example of configuration file content:

```
#CONFIGURATION NAME
SET WFS_PROG 2
SET CCD_BACKGROUND "CONFIGNAME_CCD_BACKGROUND.DAT"
```

...
...
SET PIXEL_LOOKUP_TABLE "CONFIGNAME_PIXEL_LOOKUP_TABLE.DAT"

The first line is just a comment then there are two single parameter definitions. The forth line is a parameter which has to be loaded from another binary file. Files are named following this base scheme:

ConfigName_ContentType.Extension

The main initialization file is simply named like ConfigName_Base.ini.

This structure has two advantages:

- The files remain easily readable by humans, editable and manageable. Configuration can be organized easily, copied and changed.
- The user can load a specific configuration through the *LD_CONFIGURATION* WCP/WIF command supplying just the main configuration file name. On the other side configuration files can be simply generated trough the *SAVE_CONFIGURATION* command.

For instance we can send the *SAVE_CONFIGURATION* command with the following parameter "SystemTest010506". The WIF will automatically generate these files:

- SystemTest010506_Base.ini
- SystemTest010506_ ccd_background.dat
- …
- SystemTest010506_pixel_lookup_table.dat

7.2.2 WIF/WCP interface

The Wavefront controller Interface (WIF) interfaces the WCP with the RTC system. It accepts commands from WCP, which is implemented on the same CPU. Commands are transferred by means of a simple VxWorks read/write driver. The WIF is a software interface that is used to transfer all control parameters to the WFP and to read back its status information. The interaction is achieved by means of a data exchange structure that relies on a real-time semaphore to synchronize the operation between WIF and WCP. The WCP can track asynchronously the command execution by polling a command status register. The communication supports transfer of both scalars and more complex data structures (e.g. matrixes).

Hereafter we report the communication structure:

```
struct wifStruct
 {
   int semID
   int cmdIndex
   int status
   void *p_data
 }
```

- `semID` is a VxWorks semaphore identifier which is used for synchronization.
- `cmdIndex` is the command ID, as reported in 9.3, refer to the WCP/WIF command list to see all available commands and details.

- `status` is the command status register; this is needed for command tracking and return information .i.e. RUNNING, SUCCES or FAILED.
- `p_data` is a pointer to a memory area. The definition of this area is related to the specific command. For instance for the Reconstruction Matrix setup command this is a pointer to a sequence of floats representing the matrix coefficients.

For communication to the MGAOS the WIF software bases on the Command Interpreter and Executor (CIE). Its main purpose is to translate high level WCP/WIF commands to the low level communication primitives according to MGP (MicroGate udp/ip Protocol) (see 7.2.2.2) A single WIF/WCP command may translate into a single MGP command or into a sequence of commands. In the latter case, the CIE generates and controls autonomously the execution of the MGP sequence. All commands to the MGAOS are sent only by the CIE, which has full control of the dedicated MGAOS-MVME6100 Ethernet link, thus the CIE can act as arbiter between real time data transfers (STRAP counts and IRIG timing information), and WIF/WCP commands.

During "operational mode" (default mode) the WIF will only accepts WIF/ WCP protocol compliant commands. For low level debugging purposes the WIF can be switched into "debugging mode". In this mode the WIF will accept also direct MGP commands from external Ethernet, and in this sense it will just act as a bridge for MGP commands. This allows the user to use other SW instruments like Microgate EngPanel or Microgate Matlab scripts, or other instruments which use the MGP protocol to communicate and control the RTC.

This feature is extensively used for system development and testing, together with a dedicated tool that allows to generate WIF/WCP commands directly from an external Ethernet interface, without requiring the WCP itself. These aspects are treated in detail in [AD7].

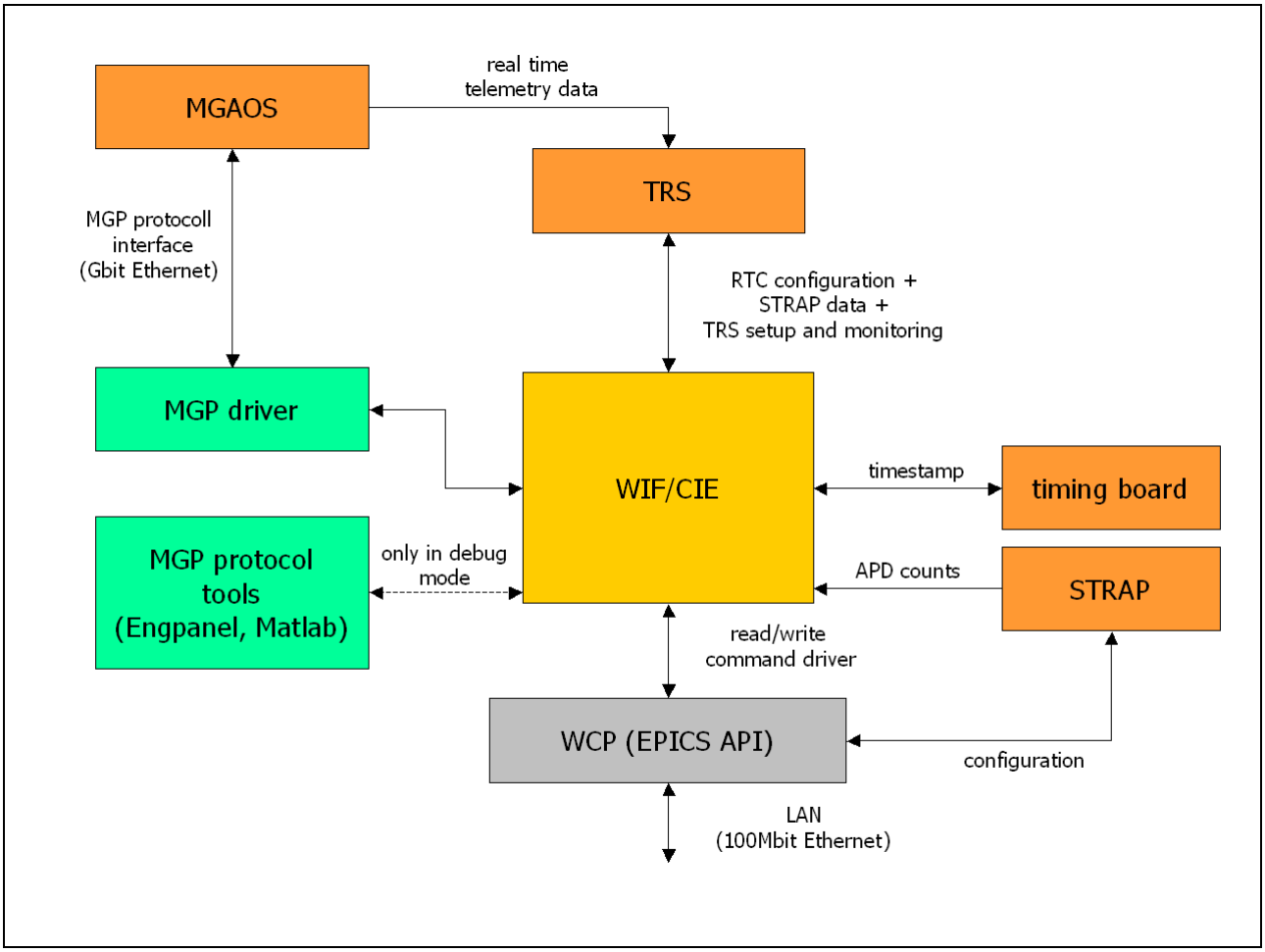


Figure 31 - MVME 6100 software block diagram

7.2.2.1.1 CIE command translation

As already said in the previous chapter, the main job of the CIE is sequencing WIF commands. There are WIF commands which are not just reading/writing parameters, some of them are more complex. For instance the STATE_CMD WIF command is used to initialize/standby the RTC. An initialize command can not be accomplished in a "single" step because there are several tasks to be performed. The CIE will threat commands which can not be accomplished with a single operation. A task will be launched to handle the command. The WCP can monitor the execution of each command by reading the status field. Each WIF/WCP command will have at least the following three possible states: RUNNING, SUCCESS and FAILED.

For certain commands, e.g. STATE_CMD, it is convenient to have more information then this, it might be convenient when RUNNING to know what the CIE is actually doing, or, if it fails, the exit status should be known. The general rule is that if the status field of a command is:

- > 0 the command is running, the value indicates what it is doing (1=RUNNING general information, job not specified).
- = 0 the command has successfully finished
- < 0 the command has terminated due to an error. The value gives more information on the error occurred (-1=FAILED general information, reason not specified)

Following is an example of the INIT command CIE execution:

```
cmd.status=LOAD_SYS_PAR
if (res=loadSysPar()<0)
   cmd.status=res;
   return;
cmd.status=INIT_WFS
if (res=initializeWFS()<0)
   cmd.status=res;
   return;
cmd.status=INIT_DM
if (res=initializeDM()<=0)
   cmd.status=res;
   return;
cmd.status=INIT_DTT
if (res=initializeDTT()<0)
   cmd.status=res;
   return;
cmd.status=INIT_UTT
if (res=initializeUTT()<0)
   cmd.status=res;
   return;
```

Hereafter we expand one of the above functions:

```
function res initializeDTT()
   cmd.status=POWER_UP
   openHVRelay()
   enableHVSupply()
   setHVOutput(0)
   if HVOutput!=0
       return(ERROR_HV_OUTPUT_NOT_ZERO)
   closeHVRelay()
   cmd.status=RAMP_MIRROR_FLAT
   rampMirrorToFlatPosition()
   if (checkPosition()!=OK)
       return(ERROR_POSITION)
   if (checkVoltage()!=OK)
       return(ERROR_VOLTAGE)
   if clmp
       cmd.status=CLOSE_MP
       setClmpGain(0)
       closeClmpGain()
       rampClmpGain(1)
   return(0)
```

The defined WCP/WIF interface as shown above allows a convenient command tracking mechanism. All command errors and states varies from command to command. The complete list of handled errors and intermediate command states will be generated during final implementation of the code.

7.2.2.1.2 Heartbeat/health monitoring

Periodically with a frequency of at least 10 Hz, the WCP must read the RTC status by means of the GET_STATUS command. When the MGAOS gets this particular command it will clear an internal Watchdog. This Watchdog mechanism assures that the master, i.e. the WCP, is always online and operation is normal. If there is some communication problem with the WCP, the Watchdog will trigger a mirror software shutdown procedure, and after few seconds (equal to mirror shutdown time, TBD during development) it will also reset itself.

The same mechanism is safe also when the MGAOS itself is failing. In fact, if the MVME does not receive correct replies to the status polling, it will suspend the status check thus causing a reset of MGAOS.

The way the Watchdog is implemented is crucial for correct operation of the above described mechanism.

The Watchdog is realized in Hardware (BCU logic) and before resetting the whole MGAOS it sends a hardware signal on the MGAOS backplane. This signal will start the safe shutdown procedures on the DSP and HVC boards.

7.2.2.2 MGP driver

MGP driver is a set of commands to operate the MGAOS system through Ethernet. The MGP commands are designed to allow a low level access to the MGAOS system. Read and write operations can be performed from/to each MGAOS component, i.e. SRAM, SDRAM, DSP RAM, FLASH, of each board. Instead of sending real operational commands, MGAOS operation is controlled by changing the status of specific memory locations, checked by the MGAOS software.

The single commands are specified by the MGP-protocol whose packets are a subset of the IP/UDP protocol. According to the standard Ethernet protocol, the structure of the Ethernet packet used for the MGAOS communication is the following:

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 communication 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 40 – MGP record structure

The "data…" field of the UDP/MGP protocol is composed by an header and a data buffer in according to the command specified.

The command reply is obtained by an asynchronous Ethernet command sent by the communication board to the host that has sent the command. Each command has a command identifier which is copied to the reply command to match the command sent to the reply received.

The following table describe the 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 the 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 the communication board to the host); bit 1 = as a quad word (to enable the burst write transaction writing to the DSP).
	command identifier	8	a 8 bit identifier of the packet. The same value is used on the reply packet to match the sent command to the reply
header dword #2	start address	32	start address where to write or to read the data
Data…	…		if necessary

Table 41 - MGP packet structure

The first DSP and last DSP are used to define from which boards to which board the command should be performed. It reduce a lot the Ethernet communication loading in fact it is possible, for example, write the same buffer or different buffers or read different portion of memory of several devices with only one Ethernet command.

The limitations are:

- only one device can be accessed by a single command
- all AdOpt DSP boards addressed by a broadcast command must be in consecutive sequence
- 'special' boards, like BCU and HVC, are accessed individually and not in conjunction with DSP boards

The size of a single Ethernet packet is limited by the Ethernet standard protocol. For the Fast Ethernet, maximum 100Mbit of speed, the maximum acceptable size is 1512 bytes, meanwhile for the Gigabit Ethernet it is possible go on up to 65536 bytes. Of course the best size is the right tradeoff between size, which reduces the start up delay, and latency of a single packet which is stopped until the previous one is not finished.

The maximum size currently accepted by a communication boards is 4096 bytes, which requires to enable the "Jumbo frames". Of course it is possible to use a smaller packet size, for example if the system is connected to a Fast Ethernet system.

The basic MGP-commands available are listed in following table:

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
MGP_OP_RESET_DEVICES	10	Resets independently all the devices on the MGOAS crate
MGP_OP_SERIAL_SCIMEASURE	12	Manages the dedicated SciMeasure serial port
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
MGP_OP_WR_SCIMEASURE_RAM	135	Writes to the AIA lookup table
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
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	Used to notify at the supervisor that the MGAOS has detected a warning condition
MGP_OP_CMD_DIAGNOSTIC	204	Used on the reply of a diagnostic packet
MGP_OP_CMD_NULL	255	Null command

Table 42 - MGP commands list

The complete format description of all the MGP commands is available in 9.1.

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

Command name	Value	Description
MGP_FL_WANTREPLY	0	Enable the creation of a reply command from the MGAOS. To be set also when a read command is sent.
MGP_FL_ASQUADWORD	1	Used in the write to DSP memory commands. It enables a special write mode without break during the data transfer

7.2.3 STRAP and Timing support task

Another important work done on the MVME6100 is STRAP interface and time-stamping clock synchronization.

The MGAOS has no direct interface to STRAP. STRAP data needed for DTT mirror operation will be read by the STRAP interface task running on the MVME6100. The data have real time characteristics and needs

to be transferred through MGAOS-MVME6100 Ethernet interface without big latencies. To be consistent with the timing specifications and with the performance of the other real-time components, the transfer latency should be < 200µs.

This specification can be met considering the following aspects:

- Full packet control: MGAOS-MVME6100 Ethernet interface is totally private, all packets that are transferred on this interface are controlled by the MGAOS and the MVME6100, and no other packets can transit on this interface.
- Arbiter: the CIE which acts also as the command arbiter will execute real time transfer before other commands. A WCP command can not delay any real time transfer.
- Short packet: the single maximum packet frame size is kept sufficiently small, so that the packet transfer time is always significantly lower than the maximum latency. A Gbit Ethernet interface can transfer up to 20kB in 200µs. The maximum Ethernet packet size (without using jumbo-frames) of 1514 bytes is perfectly adequate. If a particular WCP command is under execution the CIE will not wait until the command has terminated but it waits only until the single UDP-packet has been transferred.

The timing information transfers (see §7.1.8) do not have actual real-time requirements, nevertheless they are treated as real time data transfers to always assure that the information is passed before the next synchronization tick.

7.2.4 System monitoring

The MGAOS boards have on-board diagnostics that allows to keep under control several ‘system-health’ parameters. Hereafter we provide a list of these parameters organized by board:

- BCU board
    a. FPGA temperature (measured directly on chip)
    b. DSP temperature
    c. Local power supply temperature
- DSP board
    a. FPGA temperature (measured directly on chip)
    b. DSP temperature
    c. Local power supply temperature
- HVC board
    a. FPGA temperature (measured directly on chip)
    b. DSP temperature
    c. Local power supply temperature
    d. High voltage drives temperature
    e. Actual voltage on each actuator (measured independently with respect to commanded voltage)
    f. Actual voltage on positive and negative high voltage power rails

All these parameters are acquired by each AdOpt board (BCU, DSP, HVC) and updated every second. External access to these parameters is obtained through the ‘GET_MGAOS_HEALTH’ WIF/WCP command.

A dedicated task on MVME will monitor continuously and autonomously these parameters. Excessive temperatures are reported also as status flag on the system status words. If the system goes and stays in

overtemperature for a longer period (exact duration to be determined during implementation), the MVME system will shutdown the MGAOS system.

Beside the MGAOS this task will also monitor the TRS status to ensure that data are registered properly, this is necessary because MGAOS-TRS communication is only a one way communication to achieve highest possible throughput. The TRS will check data integrity and if one data-frame is corrupted this will only be marked, and a data transmission failure counter will be incremented, but it will not signal the problem directly to MGAOS, instead the problem will be detected by the monitoring task.

7.2.5 System configuration storage to TRS

In order to well interpret the diagnostic data stored on TRS it is very important to know the exact operation mode/configuration for each data frame. Obviously it is not possible to save at each real time data frame the corresponding system configuration, because this would lead to saving a huge amount of redundant data. To be remembered that system configuration includes CCD-background, reconstruction matrix, control servo parameters, and so on.

To achieve this task reducing at a minimum the stored data the following scheme has been conceived.

At system startup the full configuration set is written to the TRS. Each different configuration, within a single session, is stamped with a unique id, which is simply an incremental counter. So the first configuration written to TRS (i.e. the one loaded at boot time) will have the ID #1. The full configuration set will be uploaded to TRS only once, at start-up, afterwards, when the configuration changes only the single changed parameter will be registered. For each change the configuration-ID will be incremented.

The configuration storage and change is made by the MVME6100, but it is the MGAOS which really uses the configuration parameter set. MVME6100 doesn't know the exact frame/time when MGAOS uses a new configuration set. This is the rule of the conf-ID field in the FFB (see Table 29), it will be saved together with the other time-stamped real time data. So each data frame is associated with one unique configuration ID. Knowing this ID, it is possible to reconstruct the full configuration/setup of the system.

7.3 TRS

The TRS is responsible for data storage and queries accomplishments. The TRS is based on two main blocks the server and the disk array. The software running on the disk array system is not described in this document and can be treated as a black-box provided by the disk manufacturer. It is worth noticing here that the server sees the disk array just like a normal single big SCSI-disk.

7.3.1 TRS operating system and filesystem

The server will run Linux Red Hat operating system. A very important aspect of TRS is the file-system type used for data storage. Following is a list of various file-system types supported by Linux, which are suitable for large file management, with a short description.

- Ext2fs is the most portable native Linux file-system. Drivers and access tools for ext2fs are available in many different OSs, meaning that it is possible to access ext2fs data from many non-Linux OSs. Unfortunately, most of these tools are limited in various ways. For instance, they may be access utilities rather than true drivers, they may not work with the latest versions of ext2fs, they may be able to read but not write ext2fs. In any case using the files-system with Linux only, this is the most reliable and efficient choice.
- Ext3fs is basically just ext2fs with a journal added. As such, it's quite reliable, because of the well-tested nature of the underlying ext2fs. Ext3fs can also be read by an ext2fs driver; however, when it's mounted in this way, the journal will be ignored.

- JFS was developed in the mid-1990s for AIX by IBM, then it found its way to OS/2 and then to Linux. It's therefore well tested, although the Linux version hasn't seen much use compared to the non-Linux version or even ext3fs or ReiserFS on Linux.
- XFS, from Silicon Graphics' IRIX, is another files-system supported by Linux. SGI's XFS dates from the mid-1990s on the IRIX platform, so the file-system fundamentals are well tested. It's the most recent official addition to the Linux kernel, although it has been a fairly popular add-on for quite a while. XFS comes with more ancillary utilities than does any file-system except ext2fs and ext3fs. It also comes with native support for some advanced features, such as ACLs, that aren't as well supported on most other file-systems.
- ReiserFS was the first journaling file-system added to the Linux kernel. As such, it's seen a lot of testing and is very reliable. It was designed from the ground up as a journaling file-system for Linux, and it includes several unusual design features, such as the ability to pack small files into less disk space than is possible with many file-systems. ReiserFS is currently the least portable of the major Linux-native file-systems. There is a BeOS version, but versions for other platforms have yet to appear. ReiserFS should be avoided, if cross-platform compatibility is needed.

As already mentioned above, all these file-systems are suitable for large data size. Considering that the main access to the disk array is done by a Linux machine (the TRS server), the ext3 file-system will be used, because it is the most reliable and tested file-system on Linux Kernel, which has also journal features and some portability. Journaling is important to avoid long file system checks during start-up, in case of power failures or generally in case of improper disk ‘umounts’.

7.3.2 TRS data storage

The TRS software architecture presented at PDR was based on a custom database. The rationale of this initial choice was in the high storage throughput and in the relatively simple queries to be implemented. Standard database packages are typically quite powerful in handling complicated queries, and less performing in terms of input data streaming. Another important aspect is related to the built-in post processing capabilities. Standard products provide few mathematical functions like min, max, mean and sum, and these computations are provided just for standard data-types like floats or integers. Other functions, like rms or fft of data arrays like a CCD frame might be difficult to implement or require external post-processing.

During DDR phase we learned about the database package called PostgreSQL, a widely used and well supported open source relational database system. The main aspect capturing our attention was the unique feature of customization possibilities provided by this database. It is possible for the user to define its own functions and types. User-defined functions can be written in C.

So using PostgreSQL provides all the advantages of a SQL database, like the availability of a well defined, SQL compatible query mechanism, efficient handling of multiple client connections, query flexibility. Additionally, it allows to easily adding custom functions for mathematical processing of queried data.

Therefore, the possibility of using such a standard product instead of the originally designed custom architecture is very attractive. Some general information on PostgreSQL can be found in §9.4.

7.3.2.1 TRS performance test

7.3.2.1.1 Sustained storage throughput

The main aspect to investigate is the actual capability of handling the required data input throughput of 13MB/s. To this aim, we have carried out a performance test using the following setup:

Hardware

- Mass storage: Triton F16A Disk Array

- Server: Sun Ultra20 with single CPU (64bit Opteron processor)
- FibreChannel interface between server and disk: Qlogic 2312, 2.125 Gbit/s

The disk array and the FibreChannel interface are final choices, while the server is somehow a conservative configuration; in fact we plan to use a dual-CPU machine in the final implementation.

The mass storage is configured as RAID3, with 14 storage disks and 2 more disks for hot-swapping. The total storage capability is 5.0 TB.

Software

For testing purposes, we used a Centos Linux Box and PostgreSQL version 8.0.4. To populate the database, we developed a custom client written in C, which randomly generates data and stores them into the database. The data structure couldn't be the final one because of development time issues; indeed we used one table with two columns, one for timestamp (uint64), and one for data (byte-array). After some initial database tuning we could measure a sustained throughput of 34 MB/s, regardless of disk space usage. This makes us confident that the TRS with the PostgreSQL option will easily meet the storage requirement of 13MB/s. As a comparison, on the same HW setup, using the initially planned custom database that simply stores directly the data onto a file we reached a throughput of 70-80 MB/s. Although this is twice the PostgreSQL performance, we must consider that PostgreSQL has already a proven and reliable storing mechanism while our custom test code was really primitive, without any checks and crash save mechanism. So some performance reduction should be taken into account in a final implementation of the custom database.

In the final implementation the client we used for generating the database records for testing purposes will be used as an interface block between PostgreSQL and the data source (RTC). This has two main advantages:

- code maintainability: all the RTC code is independent from the database we use on the TRS. The whole design made during PDR on the RTC telemetry remains valid. In the future, if the database will be upgraded, changes, if any, will only apply to this TRS client adapter.
- second stage buffering: the client will provide a second stage buffer, between BCU and PostgreSQL compensating short PostgreSQL delays. This gives more robustness to the whole telemetry system real time capabilities.

7.3.2.1.2 Query performance

Another key aspect is the query performance. The TRS requirements specify a query latency of 0.1s from the query to the start of reply. To get a first feeling on the PostgreSQL performance we used initially its own shell, which connects to the database through a TCP/IP socket, exactly as the clients will do in the final configuration. This shell has a timing option which measures the query execution time. In the database, the timestamp column is configured as table_index so the searches can be done efficiently. Tests have been performed with 2 TB of data stored on the database.

An example of query test is reported hereafter:

```
SELECT id FROM dbtest WHERE id=5930917;
the result
 dbtest=# SELECT id FROM dbtest WHERE id=5930917;
   id
---------
 5930917
(1 row)
Time: 79.584 ms
```

Repeating the same query gives the following result:

```
dbtest=# SELECT id FROM dbtest WHERE id=5930917;
  id
---------
 5930917
(1 row)
Time: 0.683 ms
```

It can be noticed that the response time is consistently reduced. The reason for this improvement is in the internal caching mechanism. At the second query attempt the data were already in RAM and no disk access was needed.

A certain portion of the database is always cached, and it is always updated, depending on the query. So for instance if we search for a different ID which is near the previous one we expect quick response time. This is demonstrated by the following test:

```
dbtest=# SELECT id FROM dbtest WHERE id=5930920;
  id
---------
 5930920
(1 row)
Time: 0.660 ms
```

Doing random queries changing the searched ID we experimented response times from fractions of ms up to 80 ms, depending on the searched ID being cached or not. Due to the index the search time depends only on the cache availability rather than on the particular ID.

After this initial attempt, in order to deeper investigate the actual performance we have developed a dedicated C client which randomly generates ID (timestamps) and queries the database to retrieve the corresponding CCD data. It was difficult for us to measure exactly the time interleaving between the query reception and response begin; therefore we measured the time on the client side which includes also data transfer latency. This is clearly a conservative assumption. but the results are still meaningful, considering that we never transferred large quantities of data.

The database on which we performed the tests contains telemetry data for a time span of 50 hours equivalent to 5 nights of observation. The executed query is the following

```
SELECT img FROM dbtest WHERE id=X;
```

where X is a randomly generated number. The test clients executes on the server machine. For each test always 10 clients were running simultaneously. Only one client result is reported since all clients tests gave quite similar results. In the following graphs on the y-axis we report the query response time while on the x-axis we report the test number. Each point represents a single test. We made 100.000 queries for each test and client for a total of 1.000.000 queries for each run. For each test we report the mean response time, the percentage of tests with a response time over 150 ms, over 120 ms and over 100 ms. The worst case is when the id is randomly generated over the full database range. In this case the cache has to be swapped several times and the data has to be often completely retrieved from disk.

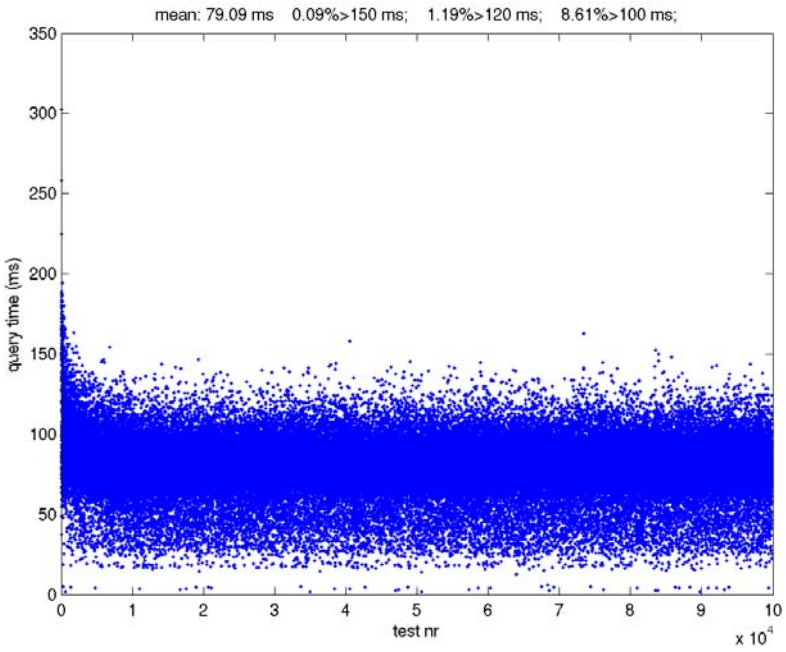


Figure 32 – Query performance on full database (5 nights span)

Afterwards, we limited the ID to a single observation night, working on the same database, i.e. it still contains all 5 nights of observation. Only the queries were limited to a single night, not the database!

The response times are shorter.

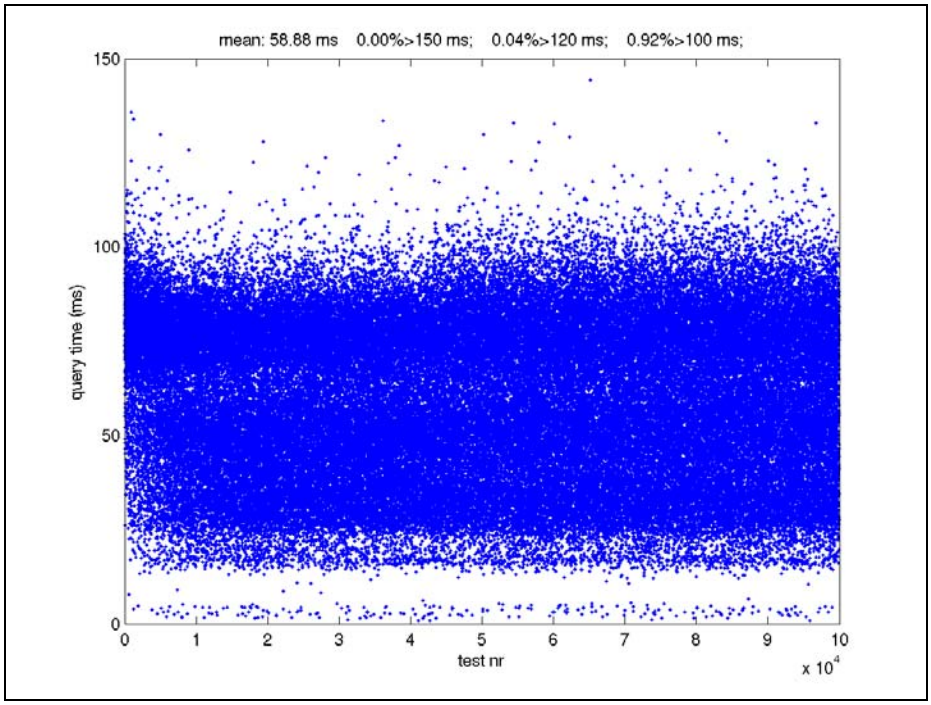


Figure 33 – Query performance on 1 night span on full database.

Here the ID is limited to 3 hours of observation

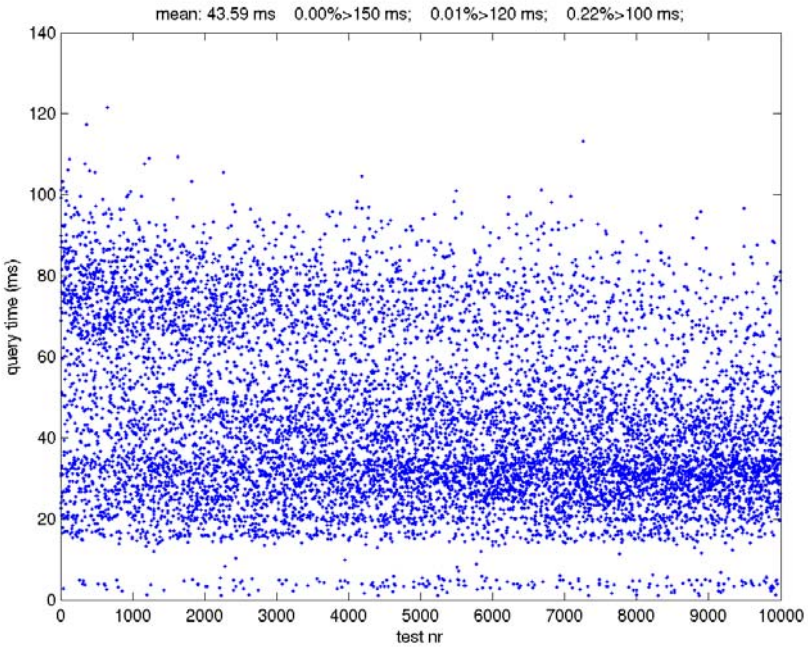


Figure 34 – Query performance on e (5 nights span)

Here the ID is limited to one single hour of observation

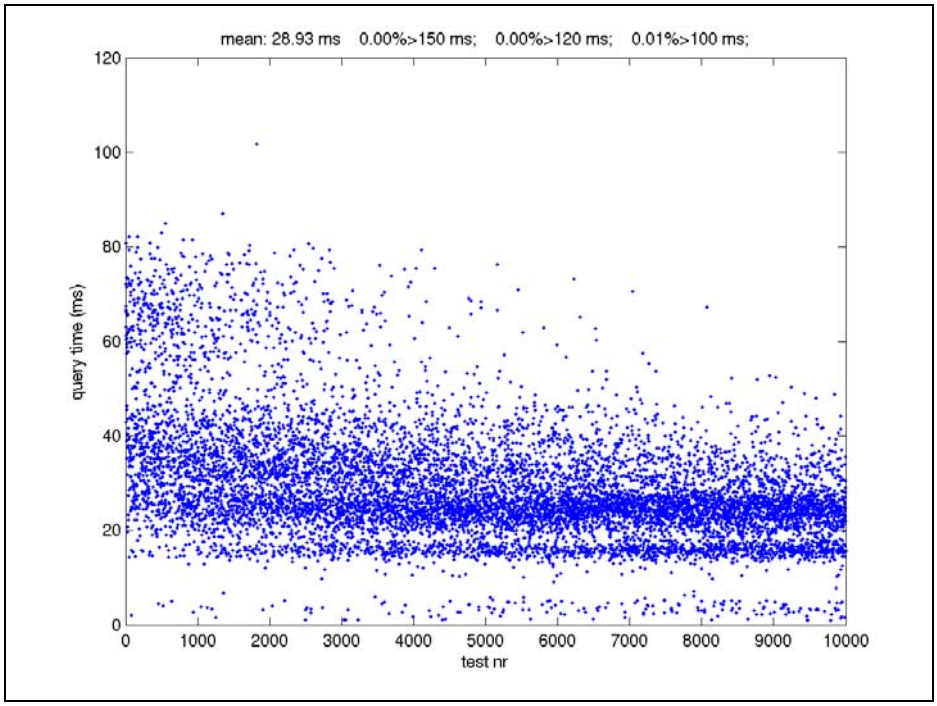


Figure 35 – Query performance on 3 hours span on full database.

7.3.2.1.3 Query performance with post-processing

Some post processing tests were also made.

Here we computed a two dimensional FFT on the CCD. The used query here was

SELECT fftCCD(img) FROM dbtest WHERE id=X;

where fftCCD is an user defined PostgreSQL extension. The size of CCD is the worst case (160x160 pixels CCID56). The ID was limited to 3 hours.

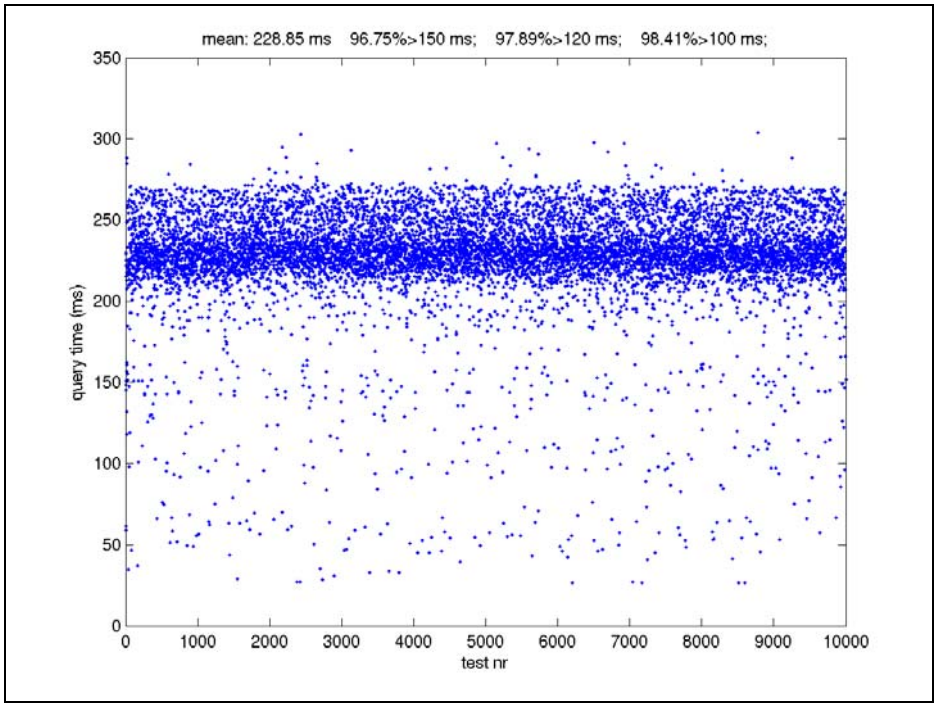


Figure 36 – Query performance on 3 hours span on full database with FFT post-processing.

Here the we computed the rms of the CCD with the following query

SELECT rms(img) FROM dbtest WHERE id=X;

It can be noticed that the response time is getting better and better due to the cache. The ID was limited to a 3 hours.

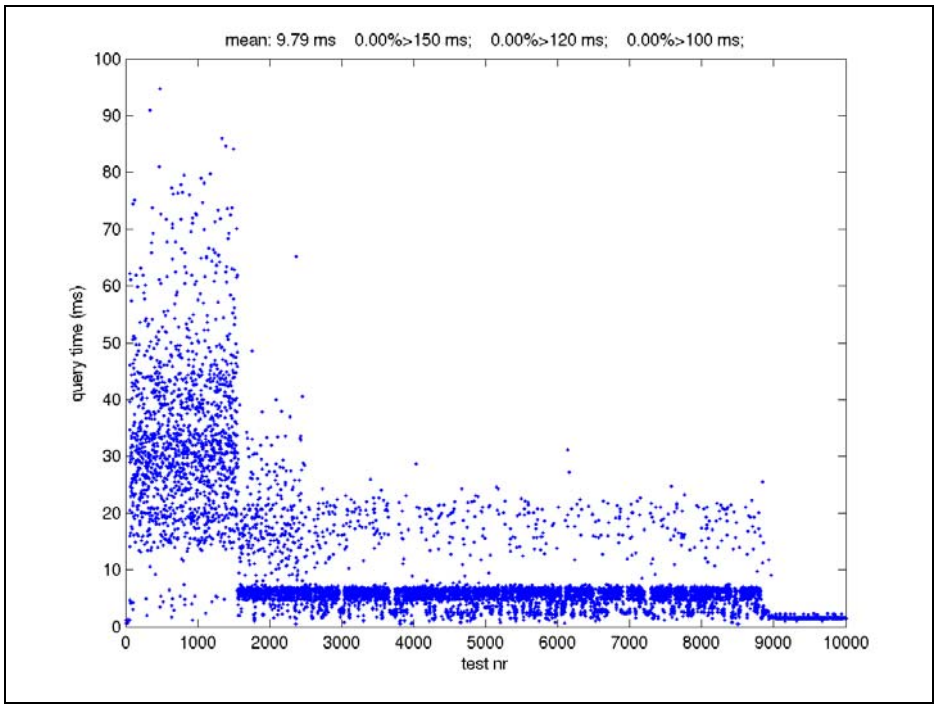


Figure 37 – Query performance on 3 hours span on full database with RMS post-processing.

The last test with RMS post processing (Figure 37) shows better performance than the pure CCD search test. The reason is always the cache. All the tests were performed in sequence so the last test benefits from the previous tests caching. This is not true for the 'first hour test' following the 'full' or the 'day' test, because in that case the cached part matches rarely. To have further confirmation on this, we repeated the 3 hour test without RMS post processing and we got better performance than before, as expected.

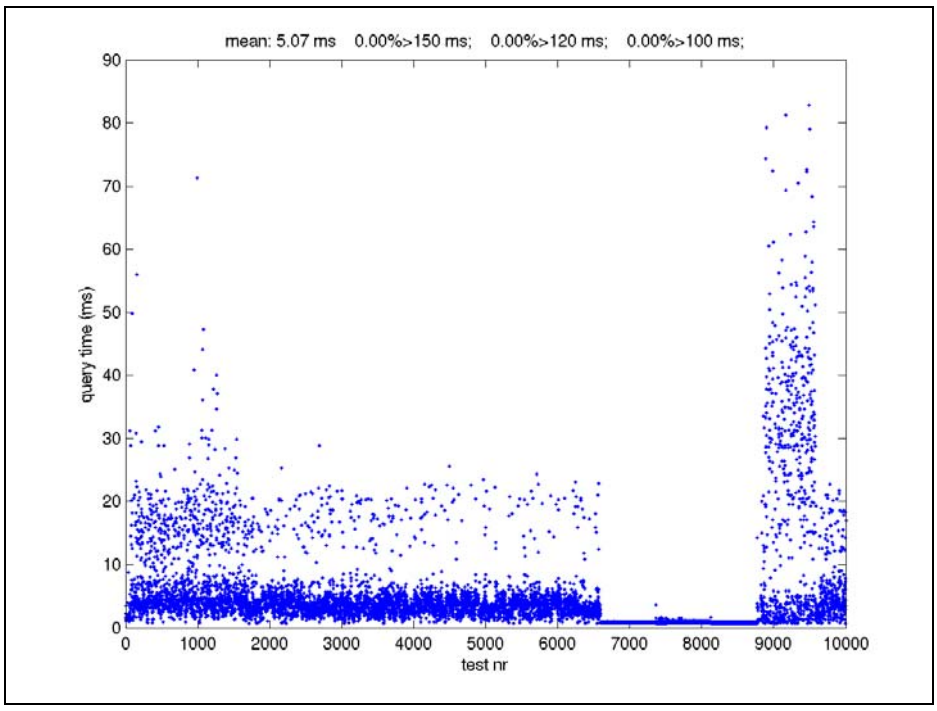


Figure 38 – Repeated query performance on 3 hours span on full database.

The query response time highly depends on the cache mechanism. If the whole data has to be retrieved completely from disk (also index) the response time might exceed the specification of 100 ms as shown in the above graphs.

Another possible concern is related to queries referring to data that have been just recorder on the database. One might argue that this data has to be cached first, before becoming 'quickly' available. This is not the way PostgreSQL operates. In fact, when a new data frame is written to PostgreSQL it also automatically cached. Therefore a query on last acquired data performs particularly well, as shown in Figure 39.

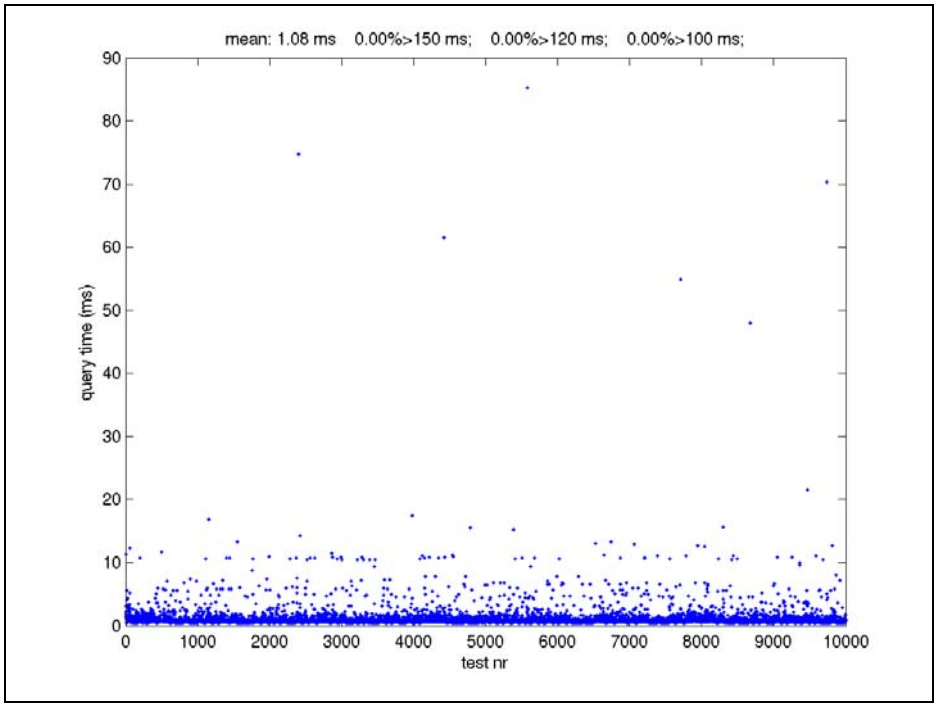


Figure 39 – Query performance on full database querying the last recorded frame.

These results were obtained without any intervention on the default settings of PostgreSQL. We are confident that there is still some improvement margin, given on one side by Hardware (the final server will be more performing, thanks to the availability of 64bit PCI instead of the 32bit PCI bus and to the second CPU), on the other side by PostgreSQL tuning. Anyway, the cache mechanism of PostgreSQL is not trivial to tune. The behavior is a result of PostgreSQL caching and of the Linux swapping mechanism.

The TRS is not a real time machine, thus the response time is never 100% deterministic, but this is true also with a custom database. The uncertainty is given by the interaction of all processes running on the server, in our case we used a standard Linux installation including the graphical environment which was also running during tests. In the final version of the TRS all software which is not needed for operation like graphical environment will be inactive.

Drawing a conclusion from these results, we can state that even if the custom solution might perform better than PostgreSQL on the pure storage throughput, the performance obtained by the standard solution exceeds by far the specification in most of the cases, even if some rare event does not reach the specification.

The advantages of using a well tested, more flexible, SQL compliant database are evident and the capability of including post-processing in the query structure is very appealing. Therefore, we are strongly in favor of the new, PostgreSQL-based architecture, and expect that the final choice will be an outcome of the forthcoming DDR meeting.

7.3.3 Server query capability

Clients will connect directly to the PostgreSQL database running on the TRS over TCP/IP socket. Libraries to create this connection are available for different programming languages. For testing purposes a PostgreSQL shell is also available.

The use of a standard SQL database greatly improves the query server capabilities. Once a client is connected to the database it can submit queries using standard SQL language.

The query capabilities also depend on the internal data organization and on the custom defined PostgreSQL extensions. In a relational database like SQL data are organized in tables where each raw represents a single record of an object and each column represents all entries for a single field. In our case the single telemetry frames will represent a row of the telemetry storage table.

TIMESTAMP	CENTROID	DTT_COMMAND	DM_COMMAND
timestamp_1	centroid_1	dtt_command_1	dm_command_1
timestamp_2	centroid_2	dtt_command_2	dm_command_2
timestamp_3	centroid_3	dtt_command_3	dm_command_3

The smallest object the SQL can operate on is a cell; e.g., it will be possible to get each individual full DM_COMMAND vector, but not a single channel out of this vector. If this is needed, it can be extracted by post-processing the data returned by SQL. This can be done by exploiting the possibility of extending PostgreSQL capabilities with a user defined function, where the channel number is passed as parameter for example:

SELECT channel(DM_COMMAND,3) FROM telemetryTbl

Another important aspect is the records selection:

Te data telemetry frames will be stored in a table where the timestamp will act as index and record ID. Queries based on index can be performed efficiently while queries where the ID is not specified will take longer to execute. As an example, a query like

SELECT max(rms(CCD)) FROM telemetryTbl WHERE timestamp > 920 and timestamp < 2343

will execute more efficiently than

SELECT max(rms(CCD)) FROM telemetryTbl

In the first case the rms(CCD) is computed only for frames with a given timestamp while in the second one the rms(CCD) is computed for all frames in the database.

Note the queries above, assumes that we have defined the RMS function for a CCD image with PostgreSQL functional extensions while the max of a float (rms(CCD)) is already a standard SQL function.

7.3.3.1 Extending PostgreSQL capability

As already mentioned above one of the most interesting feature of PostgreSQL is the possibility of extending its query capabilities by means of custom functions written in C. In this chapter the intention is to give a practical example of how this extensions works.

Our goal is to create a function, which computes the RMS of the CCD. The first task is to generate the function, hereafter we report the code:

```
1. #include "postgres.h"
```

```
2.  #include "fmgr.h"
3.
4.  PG_FUNCTION_INFO_V1(rmsCCD);
5.  Datum rmsCCD(PG_FUNCTION_ARGS)
6.  {
7.      int nr_pixels=0;
8.      float rms=0;
9.      int rmsSum=0;
10.     int i;
11.     bytea *CCD=PG_GETARG_BYTEA_P(0);
12.     nr_pixels=(VARSIZE(CCD)-VARHDRSZ)/2;
13.     for (i=0;i<nr_pixels;i++)
14.         rmsSum+=(((uint16 *)VARDATA(CCD))[i]*((uint16 *)VARDATA(CCD))[i]);
15.     PG_RETURN_FLOAT4(sqrt(rmsSum/(nr_pixels-1)));
16. }
```

The first two lines are standard includes needed for PostgreSQL function extensions.

Line 4 is a macro which must precede each function definition; the parameter is the name of the function.

Line 5 is the function declaration. Each PostgreSQL extension function has Datum as return type and PG_FUNCITON_ARGS as argument.

Line 7 to 10 are simply some variable definitions needed for the rms computation.

Line 11 declares a pointer to bytea and is initialized with the first argument passed to this function. In the specific case CCD will point to the CCD memory area. Bytea is a PostgreSQL defined type which is used for general data storage and is essentially an array of bytes.

At line 12 we compute the number of pixels, so that the function works with every CCD configuration, regardless of its size. To accomplish this job, the VARSIZE macro is available which returns the total number of bytes of the bytea memory area. From this number we must subtract VARHDRSZ which is used to store the bytea memory size. Finally we have the raw size of memory occupied by the CCD image, since each pixel is an uint16 we divide by 2 and we get the number of pixels.

Line 13 and 14 is simply the rms computation. Here the VARDATA macro returns a pointer to the data of a bytea

Line 15 we return the value which with the PG_RETURN_xxx macro.

Once we have created the function we can create a library. Assuming we saved the above function in a file named rmsCCD.c the following two commands will create the library under Linux platform. For other platforms please refer to PostgreSQL documentation:

```
gcc –fpic –c rmsCCD.c
gcc –shared –o rmsCCD.so rmsCCD.o
```

Note: pic stay's for position independent code.

Now that we have the library we must register the function into PostgreSQL database. For this task the following commands must be executed from a PostgreSQL shell:

```
CREATE FUNCTION rmsCCD(img) RETURNS float
```

```
AS 'FUNC_DIRECTORY/rmsCCD', 'rmsCCD'
LANGUAGE C STRICT;
```

FUNC_DIRECTORY/rmsCCD is the compiled library filename without the "so" extension. We need to supply also the function name this is why a complied library file could contain more than one function. The last line C STRICT is for automatically null pointer checking. If a NULL data is supplied to the function the result will be empty and there is no need to make some pointer checking within the function itself.

Once the new function is registered we can use it in the SQL language:

```
dbtest=# SELECT rmsCCD(img) FROM dbtest WHERE id=5930920;

rmsCCD
---------
 593.920
(1 row)
```

7.3.4 Data internal organization

As said in §7.3.3 the query flexibility greatly depends on how the data is organized in the database. PostgreSQL is a relational database meaning that its data organization can be represented with tables which can be interconnected. For each telemetry stream a dedicated storage table will be foreseen. Each table will have a timestamp column, which acts as index, and a column for each stream data field. Hereafter we show two table examples

FFB_tbl:

column name	column content	column type
timeStamp	timeStamp	uint64
subIntensity	Subaperture intensity vector	bytea
offsetCentroid	Offset centroid vector	bytea
residualWavefrontError	Residual wavefront error vector	bytea
residualTTFocus	Residual TT & Focus	bytea
DMActuator	DM actuator vector	bytea
residualRMS	Residual RMS	float
DTTUTTCommand	DTT and UTT mirror command data	bytea
DTTCLMPStrainGauge	DTT CLMP strain gauge	bytea
UTTCLMPStrainGauge	UTT CLMP strain gauge	bytea
confId	Conf-id	uint32

nonRTC_tbl:

column name	column content	column type

timeStamp	timeStamp	uint64
data	data	bytea

bytea is a PostgreSQL internal datatype and is essentially a memory block (sequence of bytes) and is useful for custom data representation.

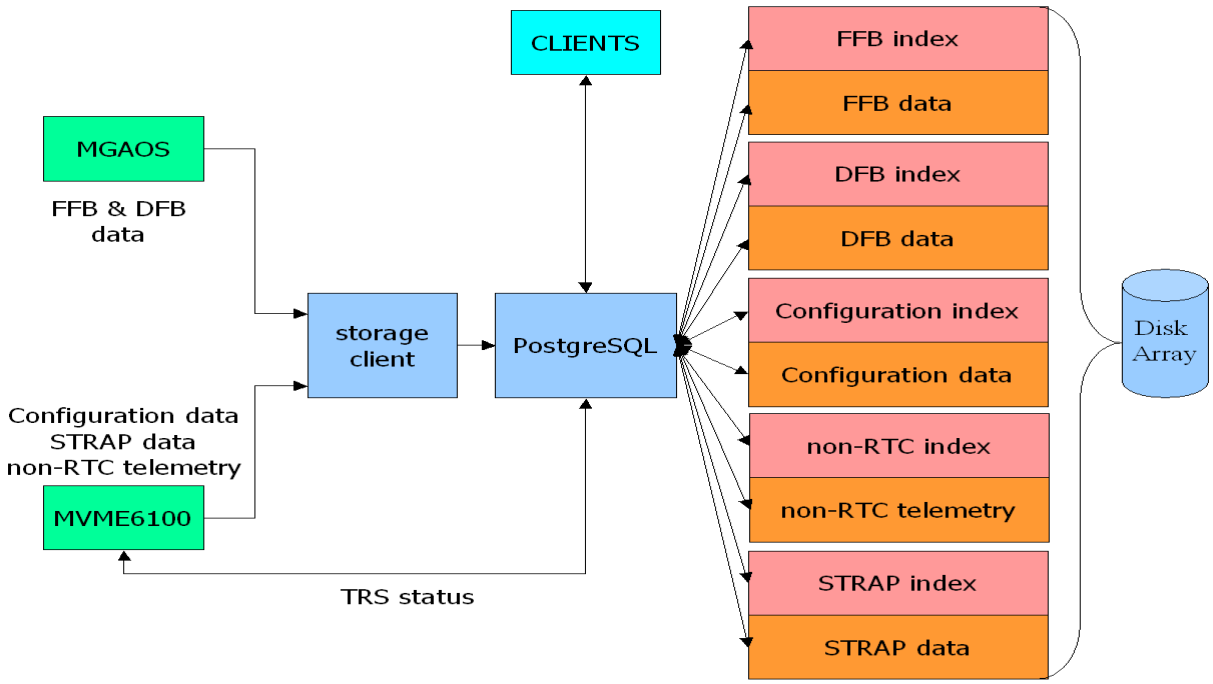


Figure 40 - TRS generic data storage

The configuration stream needs some more explanation detail. Configuration data are needed to know the RTC operating mode, this is important to correctly interpret other data. Configuration data is stored splitting information into two interconnected tables: *configuration table* and *configuration parameter table*. As already said in §7.2.5 the full configuration of the system is written only once at start-up. For subsequent changes only the single parameter is stored. For the sake of simplicity in Figure 41 we assume that the system has only 4 parameter types A, B, C and D.

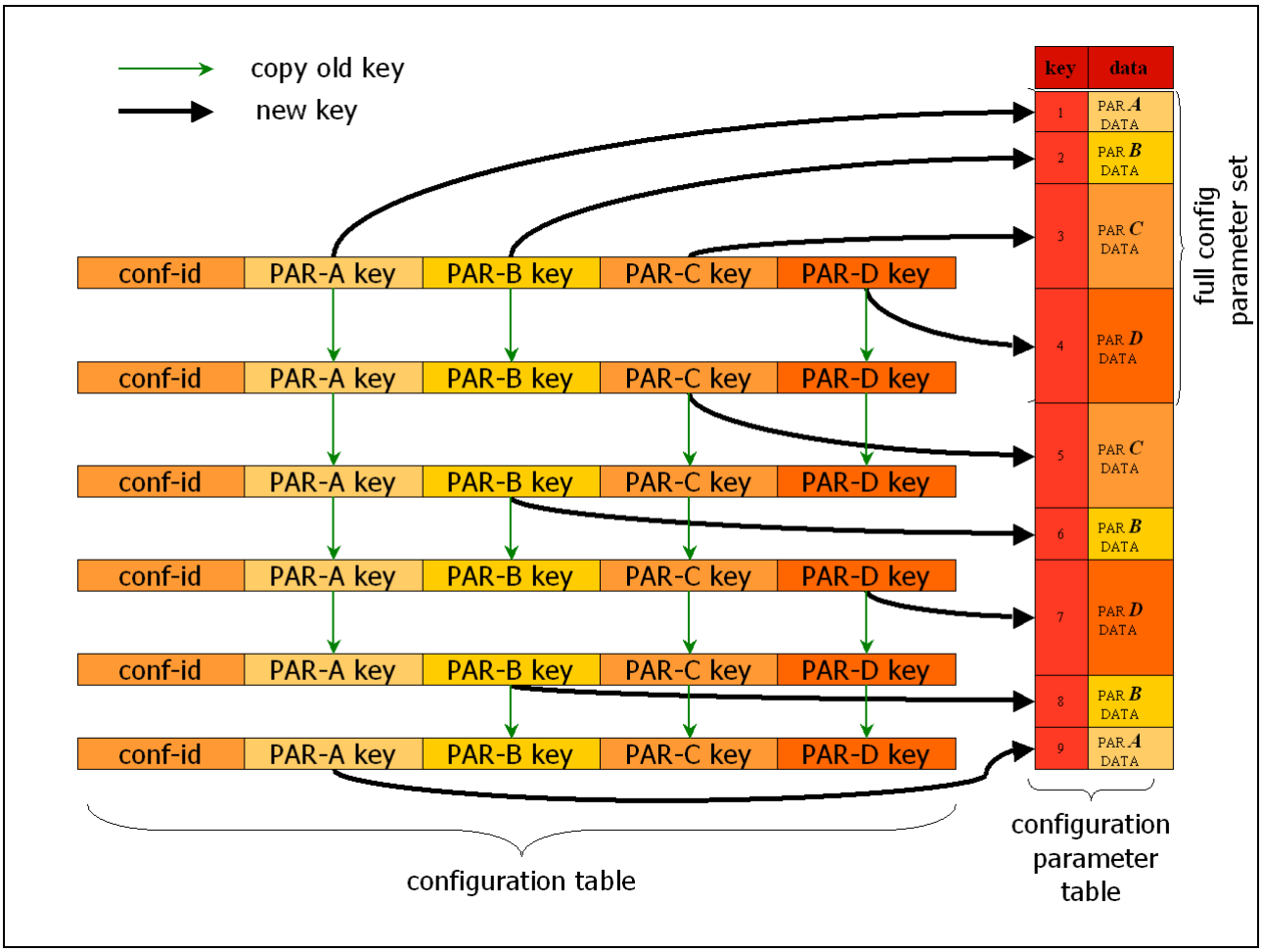


Figure 41 - Configuration data storing

These parameters must not be only single values, but can be vectors or matrices, for instance A could be the reconstruction matrix, C the background pixels and so on… The configuration tables stores the conf-id field, (which is the same stored in the FFB table for timestamping) and a key for each parameter of the system. The configuration parameter tables stores all key/parameter pairs.

At start-up all parameters are written to the data storage, the initial state is directly known. For each parameter the server generates a key. For each further single parameter change, the parameter is appended to the configuration parameter table. In the configuration table a new record is generated, all parameter keys are copied from the last record, except the one referring to the modified parameter, which is new. Doing so, it is easy to reconstruct the exact configuration of the system. Due to the fact that the configuration is known in an “incremental” way, it is very important that no changes are lost; otherwise all configuration data after the ‘missed change’ would be affected. For this reason, considering also that the configuration change rates are not that huge, an acknowledgment protocol is used for MVME6100–TRS system configuration transmission. For this purpose the TRS sends the conf-id (which is unique for each change) back to the MVME6100, confirming that he has registered that change.

7.4 SW Interfaces

7.4.1 Internal SW interfaces

Node #1	Node #2	Protocol	Description
MVME6100	MGAOS	MGP protocol (UDP socket)	WIF-WFP software interface
MVME6100	TRS	MG_DF (TCP socket port #2000)	STRAP diagnostic + configuration diagnostic
MVME6100	STRAP	Shared Ram, interrupt	STRAP can act as tip/tilt sensor
MVME6100	IRIG	Shared RAM, interrupt	Needed for timestamping
MGAOS	TRS	IP over FibreChannel with MG_DTP, MG_DF	DFB, FFB diagnostic

Table 43 - Internal SW interfaces

7.4.2 External SW interfaces

Node #1	Node #2	Protocol	Description
MVME6100	Host workstation	ftp, nfs protocol	Configuration and code download, (bootstrap)
MVME6100	WCP	read/write command driver	WCP-WIF interface
MVME6100	Workstation	Ssh (tcp/ip)	Maintenance purposes
Workstation	TRS	Dedicated Protocol	Diagnostic data evualuation/quering
MVME6100/MGAOS	Workstation	Encapsulated (in ip/udp) MGP protocol	Maintenance
MGAOS	WFS	AIA protocol	WFS interface
MGAOS	Xinteics HVA	Custom FPDP	DM command
MGAOS	UTT/DTT mirror	Dedicated protocol	UTT/DTT mirror command

Table 44 - External SW interfaces

7.5 Maintenance

7.5.1 Remote maintenance

All software maintenance work of RTC can be made remotely, with very few exceptions, for instance MVME6100 CPU board boot firmware flashing, for which a serial (VT100) interface is needed.

MGAOS is visible to the user only via Ethernet, so full remote control is guaranteed. In fact, MGAOS is not connected to a public network so direct access is not possible, but the WIF interface provides a “bridge” and software maintenance software like the Engineering Panel can be used.

The entire MGAOS maintenance can be performed remotely, in particular:

- DSP and housekeeping CPU software can be uploaded remotely through the diagnostic communication link. The housekeeping CPU software is saved on a local Flash memory

- The FPGA configuration can be completely modified remotely, via Ethernet. At next system startup, the new configuration will be loaded on the FPGA. This allows actual remote 'hardware' reconfiguration. Safety mechanisms are enforced in order to avoid dead-lock conditions. It is always possible to step back to a ‘safe’ boot configuration

MVME6100 is accessible remotely via WIF, furthermore, for maintenance, remote shell access is available. All the software is downloaded from a host workstation at boot time, so changing software is as easy as copying a file.

TRS server is accessible via ssh, all software maintenance work can be made remotely

TRS disk array is accessible via http, by means of comfortable web interface, or via telnet, all software configuration and setup can be made remotely.

7.5.2 Configuration control

The RTC software project is organized in modules. There are three main categories, operating software, driver modules, modules developed for testing purposes. Each module has a set of subdirectories which contain all the files of the module itself. Table 45 shows the list of the sub directories:

Directory name	Description
bin	this directory contains all the executables files generated by the compiler. Executables will be automatically copied here during building process.
doc	this directory contains all the document files regarding this module. The document files are simple text files which contain information about the module, memos
include	this directory contains all the header files of the module. The include files have the <.h> extension these files contain the functions prototypes, external variables reference, and class definitions;
lib	this directory contains all the complied library necessary for the module. The library files are the compiled files necessary for the linking of the executable files of the module
man	this directory contains all the man pages regarding the module functions. The man files are generated from the source using tools like doxygen. This files will be helpful for developers, they will report some reference instructions about the functions;
src	this directory contains all the source files of the module. the source files have a <.c/.cpp> extension, these files contain the C/C++ source code of the module;
test	this directory contains all the scripts for the module test.
tmp	this directory contains all the temporary files useful during the development

Table 45 - Module directories

Depending on the specific module, some directories can also be empty. Each module main directory will contain a Makefile and a Version Change Log file. The VCL contains a list of all changes made since the previous software release.

The Makefile will interpret at least following make arguments:

- clean: deletes all the temporary and executable files;
- all: compiles and links all the source files of module;
- install: copies the executable files (read from the bin directory) into the install directory which is read by the RTC during the remote (via Ethernet) booting procedure.

- *<file name>*: compiles the source coding indicated by *<file name>*

Other arguments will be provided if convenient during development.

As general software development concept, there is a file for each main class/function. Differently, the sub-functions (regarding the same argument) should be regrouped into single files. In the first case the source file name has the same name of the function, in the second case, the file name represents the common topic of the functions that it contains.

The comments are usually placed before the line of code, to remind to the reader and developer what the following line realizes.

For the entire module few include files contain all the function prototypes and the external variable reference.

All RTC software will be released following the milestones described in §5.2 of [AD3].

7.5.3 Software language

TRS, MVME6100 and NIOS II software will be implemented using C/C++ languages. For what concerns the DSP code, as mentioned in 7.1.10.1, the software will be implemented in C because of its better maintainability and readability.

ANSI C and ANSI C++ libraries will be used. The use of other third party libraries is restricted to those where source code is available and license agreements are not restrictive (GPL license preferred).

7.5.4 Development tools

7.5.4.1 HW

For hardware development the following tools will be used:

- Orcad Capture schematics entry
- Orcad Layout boards routing
- HyperLynx signal integrity simulation software
- Altrera Quartus II FPGA development tool
- Autocad

7.5.4.2 SW

For software development the following tools will be used:

- Visual DSP will be used for the MGAOS software development
- Tornado: the MVME6100 software will be developed using the Wind River Tornado environment.
- GNU ToolKit: the TRS and NIOS II software will be developed using the GNU toolkit (make, cpp, gcc, ld, as, gdb)
- Matlab 7.x: will be used to develop dedicated test and control tuning scripts.

8 DEVELOPMENT PLAN

8.1 Project aspects requiring significant hardware R&D

In this section we report the aspects of the project that require significant HW development, together with the corresponding development plan and the current status.

Description	Development plan	Status
HVC control board final design and implementation	System simulation	Done
	Bread-boarding and testing of critical parts	Done
	Final prototyping	In progress
	Software implementation	In advanced progress
	Test and calibration	To be done
FibreChannel interface between MGAOS and TRS	Design of logic level and SW level changes to current FibreChannel interface	Done
	Implementation	In advanced progress
	Test	Preliminary completed. Final test to be done.
Development of interface to Xinetics DM HVA	Circuit design	Done
	Prototyping	In progress
	Instrumental test (functional test will be possible only at the telescope)	To be done

8.2 MGAOS HW modular integration and test

After assembly, all MGAOS boards are subjected to a test and calibration procedure, including:

- Functional test
- Calibration (HVC board only)
- Storage of calibration data (HVC board only)
- Burn-in test
- Functional test after burn in
- Calibration check on sample base

8.2.1 Functional test and calibration

Functional test and calibration are performed within a single procedure, using an in-house developed automated test and calibration stand. The checks performed on the individual boards are listed hereafter:

- AdOpt BCU boards
  - On-board power supply regulators verification
  - Current absorption measurement
  - Backplane diagnostic and real-time communication test

  - On-board memories complete R/W test (Flash, SDRAM, SRAM)
  - DSP memory complete R/W test
  - External interfaces test: Ethernet, programmable digital ports
  - Functional test of on-board temperature sensors
- AdOpt DSP boards
  - On-board power supply regulators verification
  - Current absorption measurement
  - Backplane diagnostic and real-time communication test
  - On-board memories complete R/W test (Flash, SDRAM, SRAM)
  - DSPs memory complete R/W test
  - Functional test of on-board temperature sensors
- AdOpt HVC boards
  - On-board power supply regulators verification
  - Current absorption measurement
  - Backplane diagnostic and real-time communication test
  - On-board memories complete R/W test (Flash, SDRAM, SRAM)
  - DSPs memory complete R/W test
  - Functional test of on-board temperature sensors
  - Functional test of HV drives
  - Offset and linearity calibration of HV drives
  - Verification of dynamic response of HV drives
  - Offset and linearity calibration of diagnostic voltage monitors
  - Offset and linearity calibration of strain gauge sensor inputs
  - Verification of dynamic response of strain gauge sensor inputs
- Backplanes
  - Functional test
- WFS and DM interface boards
  - Functional test
- Reset board
  - Isolation test
  - Functional test

8.2.2 Burn-in tests

After functional test and calibration, all boards are subjected to a burn-in test. The purpose of this test is to put to evidence youth problems on assembled boards, in other words to exclude the first, high mortality part of the typical ‘bathtub’ failure curve.

The burn-in test of the crate components is conducted by running the system at moderately high ambient temperature (~40°C) for 24 hours. This test is expected to simulate ~4000 operating hours, corresponding to 1 year of system operation.

8.2.3 Storage of calibration data

Calibration data referring to the HVC board are permanently stored into the on-board Flash memory. The parameters are automatically loaded by the DSP software and used to adjust the ADC inputs and DAC outputs.

8.3 Software development

8.3.1 SW modular implementation and test

The software development work will follow three main branches: MGAOS SW, MVME6100 SW, TRS SW.

Table 46 shows how the software development work will be split in different stages/phases.

Development stage	**MGAOS software**	**MVME 6100 Software**	**TRS Software**
Development environment setup	VisualDSP++ development system setup Quartus development system setup	Tornado development system setup	GNU development toolkit setup
Target setup	Tools to upload DSP code, Nios code and modify logic configuration	VxWorks boot loader and base VxWorks kernel configuration	Linux operating system installation on server
Configuration, boot scripts	Comfortable boot procedures/start-up scripts will be developed for each RTC block		
Peripheral Interfaces	Dedicated FPGA module for AIA interface emulation	Irig timing board operation STRAP interface	Qlogic FibreChannel operation Disk Array configuration
Inter operation/ communication between blocks	This phase is intended to provide the low level communication between the 3 blocks which includes: • MGAOS-TRS communication. (IP over FibreChannel) • MVME6100-MGAOS communication. (MGP protocol over Ethernet) • MVME6100-TRS communication Having communication capabilities between blocks will represent a valid and useful tool for development and testing purpose in the next step.		
Main code development	WFP development: Raw Pixel Processing, Centroid computation, Residual Wavefront computation Control Law Servo Telemetry TRS transfer manager DTT/UTT controller Tests procedures	WIF, CIE, WCP-level command sequencer, TRS transfer manager, System monitor, Test procedures	Storage Manager, Query Manager, Test procedures
System, Integration	CCD39 and CCID56 interface DM interface DTT/UTT controller interface DTT/UTT controller dynamic tuning (with Matlab)	WCP interface	Client software integration
OTB integration	Final integration with optical test bench, integral test procedures, acceptance tests at Microgate		

Table 46 - Software development table

The phases are listed in time base order, each different branch can be independently developed, which guarantees a certain level of flexibility. There are stages like the intercommunication between blocks which requires synchronization between the three development branches. Although it is not obligatory to strictly follow this development plan it represents a guide to take advantage from code developed in other phases, i.e. TRS could be developed independently from the other blocks, but it could be useful for the other blocks to have some features of TRS already working.

9 Annex

9.1 MGP commands list

This annex includes the format of all MGP commands.

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 dword #0	first crate	4	0x00
	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

MGP_OP_WRSEQ_DSP		this command allows to write different buffers to all the DSPs as defined into the 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_WRSEQ_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 * (last_dsp-first_dsp+1)	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

MGP_OP_RDSEQ_DSP		this command allows to read a portion of all DSPs memory as defined into the 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_RDSEQ_DSP
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to read the data
Data…		0	not used
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	length * (last_dsp-first_dsp+1) if success otherwise 0
	flags	8	0x00
	command identifier	8	same command identifier of the master command
header dword #2	start address	32	0x00000000
Data…		length * (last_dsp-first_dsp+1) or 0	data buffer

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	0x0002
	flags	8	0x00
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	0x00000000
Data…		32 * 2	see the following table to configure the first dword. The second dword is unused and should be set to zero
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

	AdOpt BCU board	**AdOpt DSP board**	**AdOpt HVC board**
bits 0 - 1	reset all crate	not used	not used
bits 2 - 3	reset all boards excluded the BCU board	not used	not used
bits 4 - 5	not used	reset board	reset board
bits 6 - 7	reset FPGA	reset FPGA	reset FPGA
bits 8 - 9	reset FLASH	reset FLASH	reset FLASH
bits 10 - 11	reset DSP	reset DSP #0	reset DSP #0
bits 12 - 13	not used	reset DSP #1	not used
bits 14 - 15	reset Ethernet chip	not used	not used
bits 16 - 31	not used	not used	not used

The reset operations available are: 00 = do nothing;

01 = to de-assert the signal reset;

10 = generate a sequence of reset if the device was operating;

11 = asserts the signal reset.

MGP_OP_SERIAL_SCIMEASURE	this command manages the dedicated SciMeasure serial port		
	Name	Size (bit)	Value
header dword #0	first crate	4	0x00
	first DSP	8	0xFF (allowed on BCU board only)
	last crate	4	0x00
	last DSP	8	0xFF (allowed on BCU board only)
	command	8	MGP_OP_SERIAL_SCIMEASURE
header dword #1	length	16	number of bytes to send to the SciMeasure controller or 0 to read the serial status
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	0
Data…		length / 4	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	0x0002
	flags	8	0x00
	command identifier	8	same command identifier of the master command
header dword #2	start address	32	0x00000000
Data…		32 * 2	dword #0 = serial status register dword #1 = number of bytes returned by the SciMeasure controller

MGP_OP_LOCK_FLASH		this command allows to lock a portion of the Flash area to avoid the possibility to overwrite it	
	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_LOCK_FLASH
header dword #1	length	16	0x0001
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to start to lock the flash
Data…		32	size of portion of flash to lock
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

MGP_OP_UNLOCK_FLASH		this command allows to unlock the entire Flash, it is not possible to unlock a single portion of Flash	
	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_UNLOCK_FLASH
header dword #1	length	16	0x0000
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to start to lock the flash
Data…		0	not used
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

MGP_OP_CLEAR_FLASH		this command allows to clear a portion of the Flash area	
	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_CLEAR_FLASH
header dword #1	length	16	0x0001
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to start to clear the flash
Data…		32	size of portion of flash to clear
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

MGP_OP_WRITE_FLASH		this command allows to write the same buffer to all the flash as defined into the header. It is not possible to perform a write sequential to the flash device	
	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_WRITE_FLASH
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	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

MGP_OP_RDSEQ_FLASH		this command allows to read a portion of all flash memory as defined into the 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_RDSEQ_FLASH
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to read the data
Data…		0	not used
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	length * (last_dsp-first_dsp+2) / 2 if success otherwise 0
	flags	8	0x00
	command identifier	8	same command identifier of the master command
header dword #2	start address	32	0x00000000
Data…		length * (last_dsp-first_dsp+2) / 2 or 0	data buffer

MGP_OP_WR_SCIMEASURE_RAM		this command allows to write a buffer to the lookup table of the SciMeasure to BCU logic interface.	
	Name	**Size (bit)**	**Value**
header dword #0	first crate	4	0x00
	first DSP	8	0xFF (allowed on BCU board only)
	last crate	4	0x00
	last DSP	8	0xFF (allowed on BCU board only)
	command	8	MGP_OP_WRITE_FLASH
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	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

MGP_OP_CLEAR_SDRAM		this command allows to clear a portion of the Sdram area	
	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_CLEAR_SDRAM
header dword #1	length	16	0x0001
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to start to clear the Sdram
Data…		32	size of portion of Sdram to clear
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

MGP_OP_WRSAME_SDRAM		this command allows to write the same buffer to all the Sdram as defined into the 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_WRSAME_SDRAM
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	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

MGP_OP_WRSEQ_SDRAM		this command allows to write different buffers to all the Sdram as defined into the 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_WRSEQ_SDRAM
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to write the data
Data…		length * (last_dsp-first_dsp+2) / 2	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

MGP_OP_RDSEQ_SDRAM		this command allows to read a portion of all Sdram memory as defined into the 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_RDSEQ_SDRAM
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to read the data
Data…		0	not used
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	length * (last_dsp-first_dsp+2) / 2 if success otherwise 0
	flags	8	0x00
	command identifier	8	same command identifier of the master command
header dword #2	start address	32	0x00000000
Data…		length * (last_dsp-first_dsp+2) / 2 or 0	data buffer

MGP_OP_CLEAR_SRAM		this command allows to clear a portion of the Sram area	
	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_CLEAR_SRAM
header dword #1	length	16	0x0001
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to start to clear the Sram
Data…		32	size of portion of Sram to clear
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

MGP_OP_WRSAME_SRAM		this command allows to write the same buffer to all the Sram as defined into the 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_WRSAME_SRAM
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	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

MGP_OP_WRSEQ_SRAM		this command allows to write different buffers to all the Sram as defined into the 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_WRSEQ_SRAM
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to write the data
Data…		length * (last_dsp-first_dsp+2) / 2	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

MGP_OP_RDSEQ_SRAM		this command allows to read a portion of all Sram memory as defined into the 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_RDSEQ_SRAM
header dword #1	length	16	0x0000 - 0x0FC0
	flags	8	allowed command flags: MGP_FL_WANTREPLY
	command identifier	8	random value between 0x00 to 0xFF
header dword #2	start address	32	start address where to read the data
Data…		0	not used
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	length * (last_dsp-first_dsp+2) / 2 if success otherwise 0
	flags	8	0x00
	command identifier	8	same command identifier of the master command
header dword #2	start address	32	0x00000000
Data…		length * (last_dsp-first_dsp+2) / 2 or 0	data buffer

9.2 WFC C code example

In this paragraph, as an example, we present the extract of the most time critical part of the DSP code.

9.2.1 DSP code of AdOpt BCU board

This DSP executes the pixel correction and centroids computation:

```
#define SUBAP_SIZE    64 // number of pixels of each subaperture can be 4, 16, 64
#define NUM_SUBAP    304 // number of subaperture of the WFS

// wait for a new start of frame
// ...
// pixel correction and centroid computation
for (centroid_cnt=0,pixel_cnt=0;centroid_cnt<NUM_SUBAP;centroid_cnt++)
   {
   // wait for a new set of pixel to proceed with the centroids computation
   // ...
   // reset of output data
   centroid_x[centroid_cnt]=0;
   centroid_y[centroid_cnt]=0;
   intensity[centroid_cnt]=0;
   for (i1=0;i1<SUBAP_SIZE;i1+=2,pixel_cnt++)
      {
      // extract and convert pixel counts from uint16 to float format
      int_bias_pixel[0] = (bias_pixel[pixel_cnt]) & 0x0000FFFF;
      int_bias_pixel[1] = (bias_pixel[pixel_cnt]>>16);
      if (int_bias_pixel[0]&0x00008000) int_bias_pixel[0] -= 65536;
      if (int_bias_pixel[1]&0x00008000) int_bias_pixel[1] -= 65536;
      // offset pixel correction and signed adjustement
      int_corr_pixel[0] = ((raw_pixel[pixel_cnt]) & 0x0000FFFF) - int_bias_pixel[0];
      int_corr_pixel[1] = ((raw_pixel[pixel_cnt]>>16))          - int_bias_pixel[1];
      if (int_corr_pixel[0]&0x80000000) int_corr_pixel[0]=0;
      if (int_corr_pixel[1]&0x80000000) int_corr_pixel[1]=0;
      // gain pixel correction
      float_corr_pixel[0] = gain_pixel[2*pixel_cnt]   * (float)int_corr_pixel[0];
      float_corr_pixel[1] = gain_pixel[2*pixel_cnt+1] * (float)int_corr_pixel[1];
      // centroid and total flux computation
      centroid_x[centroid_cnt] += fact_x[i1]   * float_corr_pixel[0];
      centroid_y[centroid_cnt] += fact_y[i1]   * float_corr_pixel[0];
```

```
      intensity[centroid_cnt]  += float_corr_pixel[0];
      centroid_x[centroid_cnt] += fact_x[i1+1] * float_corr_pixel[1];
      centroid_y[centroid_cnt] += fact_y[i1+1] * float_corr_pixel[1];
      intensity[centroid_cnt]  += float_corr_pixel[1];
      }
   // post centroid correction
   if (intensity[centroid_cnt]<threshold_intensity)
      intensity[centroid_cnt]=threshold_intensity;
   centroid_x[centroid_cnt] /= intensity[centroid_cnt];
   centroid_y[centroid_cnt] /= intensity[centroid_cnt];
   centroid_x[centroid_cnt] *= centroid_gain_x[centroid_cnt];
   centroid_y[centroid_cnt] *= centroid_gain_y[centroid_cnt];
   centroid_x[centroid_cnt] -= centroid_offset_x[centroid_cnt];
   centroid_y[centroid_cnt] -= centroid_offset_y[centroid_cnt];
   }
// enable the DMA centroids data transfer to the DSP boards
// ...
// enable the DMA tip-tilt mirror commands data transfer to the HVC board
// ...
// polling until the residual wavefront vector is computed
// ...
// readback of the residual wavefront vector (352 elements)
// ...
// polling until the UTT mirror commands are computed
// ...
// readback of the UTT mirror commands
// ...
// averaged telemetry data computation
// ...
// ready for a new step
```

9.2.2 DSP code of AdOpt DSP board

These DSPs execute the parallel reconstructor matrix multiplier and the PID digital filter for each vector element:

```
#define NUM_CHANNELS 60  // number of channels for each DSP board
#define NUM_SUBAP    304 // number of subaperture of the WFS
#define FILTER_ORDER 3   // number of taps of PID digital filter



// wait for a new reconstructor computation
// ...
```

```
// matrix multiplier
for (i=0;i<NUM_CHANNELS;i++)
   for (i1=0,reswave_vector[i]=0;i1<2*NUM_SUBAP;i1++)
      reswave_vector[i]+=reconstructor_matrix[i1][i]*centroids_vector[i1];
// computation of last PID filter tap for each vector element
for (i=0;i<NUM_CHANNELS;i++)
   output_vector[i]=delay_output_vector[i]+ \
      reswave_vector[i]*input_filter_params[FILTER_ORDER][i];
// at this point the time critical part is finished
// update of the input / output filter time history
for (i=0;i<NUM_CHANNELS;i++)
   {
   old_reswave_vector[step][i]=reswave_vector[i];
   old_output_vector[step][i]=output_vector[i];
   }
// computation of the oldest part PID filter for the next step for each vector element
for (i=0;i<NUM_CHANNELS;i++)
   {
   delay_output_vector[i]=0;
   for (i1=step,i2=0;i1>=0;i1--,i2++)
      {
      delay_output_vector[i]+=old_reswave_vector[i1][i]*input_filter_params[i2][i];
      delay_output_vector[i]+=old_output_vector[i1][i]*output_filter_params[i2][i];
      }
   for (i1=FILTER_ORDER-1;i1>step;i1--,i2++)
      {
      delay_output_vector[i]+=old_reswave_vector[i1][i]*input_filter_params[i2][i];
      delay_output_vector[i]+=old_output_vector[i1][i]*output_filter_params[i2][i];
      }
   }
step++;
if (step==FILTER_ORDER) step=0;
// ready for a new step
```

9.3 WIF command table

Sub-system	Full Name	Function
WFS	SET_BACKGROUND	Background set
	GET_BACKGROUND	Background read
	SET_FLAT_FIELD_IMAGE	Flat field set
	GET_FLAT_FIELD_IMAGE	Flat field read
	SET_MIN_SUB_FLUX	Min subap flux set
	GET_MIN_SUB_FLUX	Min subap flux read
	SET_CENTROID_ARCSEC	Cent-to-arcsec gain set
	GET_CENTROID_ARCSEC	Cent-to-arcsec gain read
	SET_CENTROID_ORIGIN	Centroid origin set
	GET_CENTROID_ORIGIN	Centroid origin read
	SET_AVG_SUB_FLUX	Avg subap flux set
	GET_AVG_SUB_FLUX	Avg subap flux read
	SET_DEN_FREE_CENTROID	Set denominator free centroiding ON/OFF
	GET_DEN_FREE_CENTROID	Read denominator free centroiding setup
	SET_WFS_TEMP	Temp set
	GET_WFS_TEMP	Temp read
	SET_WFS_PROGRAM	Program select
	GET_WFS_PROGRAM	Program read
	SET_WFS_REPETITION_NR	Delay select
	GET_WFS_REPETITION_NR	Delay read
	SET_WFS_BIAS	Bias set
	GET_WFS_BIAS	Bias read
	SET_WFS_GAIN	Camera gain set
	GET_WFS_GAIN	Camera gain read
	SET_SUBAPERTURE_NR	Number subaps set
	GET_SUBAPERTURE_NR	Number subaps read
	SET_SUBAPERTURE_SIZE	Subap size set
	GET_SUBAPERTURE_SIZE	Subap size read
DM	SET_RECONTRUCTION_MATRIX	Reconstruction matrix set
	GET_RECONTRUCTION_MATRIX	Reconstruction matrix read
	SET_DM_SERVO_COEFFS	DM control law coefficients set
	GET_DM_SERVO_COEFFS	DM control law coefficients read
	SET_DM_ORIGIN	DM origin set
	GET_DM_ORIGIN	DM origin read
	SET_DM_LOOP	Set DM loop
	GET_DM_LOOP	Read DM loop
	SET_DM_LIMIT	Set DM limits
	GET_DM_LIMIT	Read DM limits
	SET_DM_ACTUACTOR_MAP	DM actuator map set
	GET_DM_ACTUACTOR_MAP	DM actuator map read
DTT	SET_DTT_OFFSET	DTT ctrl offset set
	GET_DTT_OFFSET	DTT ctrl offset read
	SET_DTT_SERVO_COEFFS	DTT control law coefficients set
	GET_DTT_SERVO_COEFFS	DTT control law coefficients read
	SET_DTT_ANGULAR_OFFSET	DTT mirror offset set
	GET_DTT_ANGULAR_OFFSET	DTT mirror offset read
	SET_DTT_DISTURBANCE	Set DTT disturbance vector
	GET_DTT_DISTURBANCE	Read DTT disturbance vector
	CTRL_DTT_DISTURBANCE	DTT disturbance start
	GET_DTT_DISTURBANCE_BUFFER	Read DTT disturbance buffer setup
	SET_DTT_SENSOR_INPUT	DTT sensor set

	GET_DTT_SENSOR_INPUT	DTT sensor read
	SET_DTT_SERVO_LOOP	Set DTT loop
	GET_DTT_SERVO_LOOP	Read DTT loop
	SET_DTT_CLMP_LOOP	Set CLMP loop
	GET_DTT_CLMP_LOOP	Read CLMP loop
	SET_DTT_CLMP_REC_MATRIX	CLMP recon mat set
	GET_DTT_CLMP_REC_MATRIX	CLMP recon mat read
	SET_DTT_CLMP_SERVO_COEFFS	CLMP control law coefficients set
	GET_DTT_CLMP_SERVO_COEFFS	CLMP control law coefficients read
	SET_DTT_LIMIT	Set DTT limits
	GET_DTT_LIMIT	Read DTT limits
	SET_SKY_BACKGROUND	Set sky background
	GET_SKY_BACKGROUND	Read sky background
	SET_FLUX_TRESHOLD	Set flux treshold
	GET_FLUX_TRESHOLD	Read flux treshold
	SET_INTERACTION_MATRIX	Set interaction matrix
	GET_INTERACTION_MATRIX	Read interaction matrix
UTT	SET_UTT_OFFSET	UTT ctrl offset set
	GET_UTT_OFFSET	UTT ctrl offset read
	SET_UTT_SERVO_COEFFS	UTT control law coefficients set
	GET_UTT_SERVO_COEFFS	UTT control law coefficients read
	SET_UTT_ANGULAR_OFFSET	UTT mirror offset set
	GET_UTT_ANGULAR_OFFSET	UTT mirror offset read
	SET_UTT_DISTURBANCE	Set UTT disturbance vector load
	GET_UTT_DISTURBANCE	Read UTT disturbance vector load
	CTRL_UTT_DISTURBANCE	UTT disturbance start
	GET_UTT_DISTURBANCE_BUFFER	Read UTT disturbance buffer setup
	SET_UTT_SENSOR_INPUT	UTT sensor set
	GET_UTT_SENSOR_INPUT	UTT sensor read
	SET_UTT_SERVO_LOOP	Set UTT loop
	GET_UTT_SERVO_LOOP	Read UTT loop
	SET_UTT_ROTATION_ANGLE	Set UTT rotation angle
	GET_UTT_ROTATION_ANGLE	Read UTT rotation angle
	SET_UTT_CLMP_LOOP	Set CLMP loop
	GET_UTT_CLMP_LOOP	Read CLMP loop
	SET_UTT_CLMP_REC_MATRIX	CLMP recon mat set
	GET_UTT_CLMP_REC_MATRIX	CLMP recon mat read
	SET_UTT_CLMP_SERVO_COEFFS	CLMP control law coefficients set
	GET_UTT_CLMP_SERVO_COEFFS	CLMP control law coefficients read
	SET_UTT_LIMIT	Set UTT limits
	GET_UTT_LIMIT	Read UTT limits
	SET_UTT_FIXED_ACT	Set the third fixed actuator voltage (V)
	GET_UTT_FIXED_ACT	Read the third fixed actuator voltage (V)
WCP/WIF	RESET	Reset
	GET_STATUS	Status
	SET_DEBUG_MGP	Enable or disable MGP commands
	GET_DEBUG_MGP	Read MGP commands driver status
	GET_MGAOS_HEALTH	MGAOS temperatures and HV voltages
	STATE_CMD	Change State
	LD_CONFIGURATION	Load from config file
	SAVE_CONFIGURATION	Save to config file
TELEM	SET_AVG_INTERVAL_CONT	Set AVG interval for real time telemetry (continous stream)
	GET_AVG_INTERVAL_CONT	Read AVG interval of real time telemetry (continous stream)
	GET_CONT_AVG_DATA	Read averaged real time telemetry data (continuos stream)
	SET_AVG_INTERVAL_DEM	Set AVG interval for real time telemetry (on

	GET_AVG_INTERVAL_DEM	demand stream) Read AVG interval of real time telemetry (on demand stream)
	GET_AVG_RAW_CCD	Read averaged raw CCD on demand
HVC	GET_HVC_VOLTAGE_MON	Read HVC voltages

Table 47 – WIF/WCP command list

9.4 PostgreSQL

PostgreSQL is a powerful, open source relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, SunOS, Tru64), BeOS, and Windows. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL92 and SQL99 data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, Perl, Python, Ruby, Tcl, ODBC.

An enterprise class database, PostgreSQL boasts sophisticated features such as the Multi-Version Concurrency Control (MVCC), point in time recovery, tablespaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer, and write ahead log for fault tolerance. It is highly scalable both in sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL systems in production environments that manage in excess of 4 terabytes of data. Some general PostgreSQL limits are included in the table below.

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

SQL implementation conforms to the ANSI-SQL 92/99 standards. It has full support for subqueries (including subselects in the FROM clause), read-committed and serializable transaction isolation levels. And while PostgreSQL has a fully relational system catalog which itself supports multiple schemas per database, its catalog is also accessible through the Information Schema as defined in the SQL standard.

PostgreSQL has won praise from it's users and industry recognition, including the Linux New Media Award for TRS data storage organizationBest Database System and three time winner of the The Linux Journal Editors' Choice Award for best DBMS.

PostgreSQL's source code is available under the most liberal open source license: the BSD license. This license gives you the freedom to use, modify and distribute PostgreSQL in any form you like, open or closed source. Any modifications, enhancements, or changes you make are yours to do with as you please. As such, PostgreSQL is not only a powerful database system capable of running the enterprise, it is a development platform upon which to develop in-house, web, or commercial software products that require a capable RDBMS.

To learn more about PostgreSQL please visit the official PostgreSQL website at www.postgresql.org

9.5 Final COTS selection

Qty	Component	Product	Manufacturer
1	VME CPU board	MVME6100-0173	Motorola
1	IRIG board	TTM635VME-VCXO	Symmetricom
1	VME/MGAOS PSU	CPCI-AC-6U-400	Schroff
3	HV PSU	ZUP 80-2.5	Lambda
1	TRS Server	Fire x4100	Sun
1	TRS Disk Array	SR-TRITON16FA	Partners
2	Fibre Channel Communication	QLA2340-CK	Qlogic
1	VME Crate	Rackpack/8	Powerbridge

Table 48 –COTS product list

9.6 Microgate boards schematics, layout and bill of material

9.6.1 AdOpt BCU 4.0

Schematics

AdOpt-BCU-Main 4.0

AdOpt-BCU-Bus Interface 4.0

AdOpt-BCU-Clock 4.0

AdOpt-BCU-Dsp 4.0

AdOpt-BCU-Fast Link A 4.0

AdOpt-BCU-Fast Link B 4.0

AdOpt-BCU-Fast Link C 4.0

AdOpt-BCU-Fast Link D 4.0

AdOpt-BCU-Flash Sram Cpld 4.0

AdOpt-BCU-Fpga 4.0

AdOpt-BCU-Pci 4.0

AdOpt-BCU-Pio 4.0

AdOpt-BCU-Power 4.0

AdOpt-BCU-Sdram 4.0

AdOpt-BCU-Serial Jtag 4.0

AdOpt-BCU-Thermic 4.0

Layout

AdOpt-BCU-Layout TOP 4.0

AdOpt-BCU-Layout BOT 4.0

Bill of Material

AdOpt-BCU-BOM 4.0

9.6.2 AdOpt Ethernet PCI 1.1

Schematics

AdOpt-PCI 1.1

Layout

AdOpt-PCI-Layout BOT 1.1

Bill of Material

AdOpt-PCI-BOM 1.1

9.6.3 AdOpt Fastlink 2.0

Schematics

AdOpt-Fast Link 2.0

Layout

AdOpt-Fast Link-Layout BOT 2.0

Bill of Material

AdOpt-Fast Link-BOM 2.0

9.6.4 AdOpt DSP Digital Only 3.1

Schematics

AdOpt-DSP-DIGITAL ONLY- Main 3_1

AdOpt-DSP-DIGITAL ONLY- Bus Interface 3_1

AdOpt-DSP-DIGITAL ONLY- Clock 3_1

AdOpt-DSP-DIGITAL ONLY- Flash Sram Cpld 3_1

AdOpt-DSP-DIGITAL ONLY- Diagnostic 3_1

AdOpt-DSP-DIGITAL ONLY- Dsp A 3_1

AdOpt-DSP-DIGITAL ONLY- Dsp B 3_1

AdOpt-DSP-DIGITAL ONLY- Fpga 3_1

AdOpt-DSP-DIGITAL ONLY- Power 3_1

AdOpt-DSP-DIGITAL ONLY- Sdram 3_1

AdOpt-DSP-DIGITAL ONLY- Serial Jtag 3_1

Layout

AdOpt-DSP-DIGITAL ONLY- Layout TOP 3_1

AdOpt-DSP-DIGITAL ONLY- Layout BOT 3_1

Bill of Material

AdOpt-DSP-DIGITAL ONLY- BOM 3_1

9.6.5 AdOpt DSP HVC 3.2

Schematics

AdOpt-DSP-HVC- Main 3_2

AdOpt-DSP-HVC- ADC 3_2

AdOpt-DSP-HVC- Bus Interface 3_2

AdOpt-DSP-HVC- Clock 3_2

AdOpt-DSP-HVC- Coil Connector 3_2

AdOpt-DSP-HVC- Flash Sram Cpld 3_2

AdOpt-DSP-HVC- DAC 3_2

AdOpt-DSP-HVC- Diagnostic 3_2

AdOpt-DSP-HVC- Dsp A 3_2

AdOpt-DSP-HVC- Fpga 3_2

AdOpt-DSP-HVC- Power 3_2

AdOpt-DSP-HVC- Sdram 3_2

AdOpt-DSP-HVC- Serial Jtag 3_2

AdOpt-DSP-HVC- Signal Generator 3_2

Layout

AdOpt-DSP-HVC- Layout TOP 3_2

AdOpt-DSP-HVC- Layout BOT 3_2

Bill of Material

AdOpt-DSP-HVC- BOM 3_2

9.6.6 AdOpt HVC 1.0

Schematics

AdOpt-HVC 1.0

Layout

AdOpt-HVC-Layout TOP 1.0

AdOpt-HVC-Layout BOT 1.0

Bill of Material

AdOpt-HVC-BOM 1.0

9.6.7 AdOpt AIA to PIO 2.0

Schematics

AdOpt-AIA to PIO 2.0

Layout

AdOpt-AIA to PIO- Layout TOP 2_0

AdOpt-AIA to PIO- Layout BOT 2_0

Bill of Material

AdOpt-AIA to PIO-BOM 2.0

9.6.8 AdOpt PIO to DM 1.0

Schematics

AdOpt-PIO to DM 1.0

Layout

AdOpt-PIO to DM- Layout TOP 1_0

Bill of Material

AdOpt-PIO to DM-BOM 2.0

9.6.9 AdOpt Reset Board 1.0

Schematics

AdOpt-Reset Board 1.0

Bill of Material

AdOpt-Reset Board- BOM 1.0

9.6.10 AdOpt Backplane 2.1

Schematics

AdOpt-Backplane- Slot 0 2.1

AdOpt-Backplane- Connectors 2.1

Layout

AdOpt-Backplane- Layout TOP 2_1

AdOpt-Backplane- Layout BOT 2_1

Bill of Material

AdOpt-Backplane- BOM 2.1