NRC·CNRC
From Discovery to Innovation...

Science at work for Canada

Real-time Controller (RTC) Template and Gemini North Adaptive Optics Real-time Controller (GNAO RTC) using HERZBERG EXTENSIBLE ADAPTIVE REAL-TIME CONTROLLER (HEART)

Design Description Document

National Research Council Canada
Herzberg Astronomy and Astrophysics
February 2021

Table of Contents

1. INTRODUCTION

The Herzberg Extensible Adaptive Real-Time controller (HEART) is a collection of libraries and other software that can be used to control different types of Adaptive Optics (AO) systems. HEART, with customization, can support many different types of AO Systems, multi-conjugate AO (MCAO) mode, Ground Layer AO (GLAO), and Multi-Object AO (MOAO) necessitating a highly modular software architecture. Details of the HEART design can be found in [RD1]. HEART performs wavefront correction using Deformable Mirrors (DMs) and Tip/Tilt Stages (TTS). Pixels can be received from Laser Guide Star (LGS) wavefront sensors (WFS), high-order Natural Guide Star (NGS) Wavefront Sensors, On-Instrument WFSs (OIWFS) that are located in the science instruments, and on-detector guide windows (ODGW) from science imagers. These inputs are processed in real-time by HEART to compute commands to the configure DMs and to the TTS, as well as offloading information to selected mechanisms in the RTC in the Telescope, and in the client instruments. HEART is used to assemble the Template RTC and GNAO Template RTC.

When Gemini released the RFP for this project it was a large complex project, producing a system that was extremely flexible and in an extremely short amount of time. At the time HAA has just gone through a 6 year design phase for an MCAO RTC for TMT, and already started the Agile build phase for the RTC. A significant amount of time went into that design and a lot of learning lessons. There was a lag of about a year before the Agile build phase began and during that time HAA recognized that with some modifications this could easily be designed modularly and be able to support many different types of AO systems and that was the birth of HEART. HAA proposed to Gemini that we would use the HEART design that we were already building as the building blocks and perform a pre-build phase on the customizations required for the Template and GNAO RTC. This was a model that could work given the very tight schedule – full design and build of two RTC's within 25 months. This also fit well with our Agile build approach that was an incremental and iterative approach with heavy client involvement.

The pre-Build phase included an Initial Engineering Phase of ~1.5 months which was focused mainly on the requirements. The Preliminary Design Phase was 3 months which included an early Architectural design review to confirm that the design is adaptable to the required implementations, supports the requirements and to identify any required modifications. The preliminary review followed which focused on the design as developed, critical risk areas, and those requiring decisions. This final phase is the Critical Design Phase of about 2.25 months has focused on the design and any remaining critical risk areas. The design and build of the GNAO instrument has been delayed, and during the Critical Design Phase GNAO was changed from an MCAO RTC to GLAO/SCAO with the Template RTC unchanged.

During the build phase, which started at the beginning of the contract, the focus was on the HEART implementation and building framework to support the design. The Agile build development strategy has seen a 4-week iteration cycle where tests are written at the same time as the code. Following CDR, the builds will also include the custom functionality required for the Template and GNAO Template RTC. During the entire build, the code will be delivered to Gemini and, upon receiving the code, they will review and run the test suites that are delivered.

The Factory Acceptance Test (FAT), includes re-executing of all built in tests written during development and confirmation that the RTC functionality is delivered through component and integration tests. When the GNAO instrument is available and the RTC equipment has been integrated, an onsite-acceptance test will be performed.

1.1 PURPOSE

The purpose of this document is to describe the Template RTC and GNAO Template RTC and to show how the design satisfies the requirements. This document will discuss how HEART is used and identify the parts of HEART that will be customized for Gemini. It will also demonstrate how the HEART is customized to produce the two RTC's.

The Template RTC is a MCAO implementation, 5 Laser Guide Stars (LGSs) and 3 Deformable Mirrors (DMs) with the inputs and outputs simulated and suitable for experimentation in association with end-to-end AO simulation software. The Template RTC will be reconfigured and extended to support hardware interfaces to the future Gemini North Adaptive Optics (GNAO) facility, the GNAO Template.

The intended audience of this document is future Gemini AO System designers, to provide them with insight into the design of the Template RTC and GNAO Template RTC using HEART.

The roadmap in Section 2.4 will guide the reader through the SOW requirements for the Critical Design Phase of the RTC's.

This document provides the following information:

- Section 2 gives an overview of the Template RTC and the GNAO Template RTC.
- Section 3 addresses the modifications done since Preliminary Design.
- Section 4 is a description of all major software components.
- Section 5 is a more in-depth look at the RTC Software components.
- Section 6 describes the Agile process used along with addressing the milestones.
- Section 7 addresses the risks, design trades, and future phase plans.
- Section 8 has the technical performance metrics along with the timing budget.
- Section 9 describes the Template RTC software components needed to create the RTC.
- Section 10 describes the GNAO Template RTC software components need to create the RTC.
- Section 11 discusses the hardware required to realize both RTC's, including the interfaces.
- Section 12 gives an overview of the custom parts of the observatory interface.
- Section 13 gives an explanation of how the RTC would be controlled in a typical observing scenario.
- Section 14 gives the relevant performance benchmarking results, including recent benchmarking and discusses how these relate to the requirements for Gemini.

1.2 SCOPE

This document will only address the Template RTC and GNAO Template architecture using HEART and custom parts of HEART. In some cases it may be necessary to give an overview of standard HEART functionality to make the design clear. HEART design is detailed in [RD1].

1.3 REFERENCE DOCUMENTS

RD1 HEART Adaptive Optics Design Description Document

RD2 HEART External Interface Definition Document

RD3 GNAO Telemetry Storage Details

RD4 GNAO Requirements

RD5 GNAO RTC FAT Plan

RD6 Risk Register

RD7 Development Process Document

RD8 GNAO Real-time Controller Management Plan

RD9 SPIE Paper on Thirty Meter Telescope Narrow Field InfraRed Adaptive Optics System Real-Time Controller Prototyping Results

RD10 Onsite Assembly and Installation (OAI) plan

RD11 On-site Acceptance Test (OAT) plan

RD12 GNAO Schedule

RD13 GNAO Template RTC ICD

RD14 HEART Internal Interface Definition Document

1.4 CHANGE RECORD

Revision	Date	Section	Modifications
DRF01	2020-08-06	All	Initial draft
REL01	2020-11-19	All	Initial release
REL02	2021-01-04	All	Updates from PDR comments
REL03	2021-02-12	All	Updates for CDR

1.5 ABBREVIATIONS

CB – Circular Buffer

DM – Deformable Mirror

DRD – Design Requirements Document

FAT – Factory Acceptance Test

FOV – Field of View

FSM – Fast Steering Mirror

GAO – Gemini Adaptive Optics

GLAO – Ground Layer Adaptive Optics

GNAO – Gemini North Adaptive Optics

HEART – Herzberg Extensible Adaptive Real-time Controller

HO – High Order

HOT – High Order Truth

HRT – Hard Real-Time

ICD – Interface Control Document

LGS – Laser Guide Star

LO – Low Order

LOT – Low Order Truth

MCAO – Multi-Conjugate Adaptive Optics

N/A – Not Applicable

NGS – Natural Guide Star

OL – Open Loop

OOMAO – Object-Oriented, Matlab & Adaptive Optics

POL – Pseudo Open Loop

POLC – Pseudo Open Loop Control

PWFS – Pyramid Wavefront Sensor

RTC – Real-time Controller

SCAO – Single Conjugate Adaptive Optics

SCS – Secondary Control System

SFS – Slow Focus Sensor

SOW – Statement of Work

SRT – Soft Real-Time

TBC – This item still needs to be confirmed

TBD – This item still needs to be determined

TPM – Technical Performance Metrics

TTS – Tip/Tilt Stage

WC – Wavefront Corrector

WFS – Wavefront Sensor

YAO – Yorick Adaptive Optics

2. TEMPLATE RTC AND GNAO TEMPLATE RTC OVERVIEW

2.1 OVERVIEW

The Herzberg Extensible Adaptive Real-time Toolkit (HEART) is a CPU-based software framework for the development of a Real-Time Controller (RTC) for a generalized Adaptive Optics (AO) system and is discussed in more detail in [RD1]. The building blocks of HEART are used to create an AO RTC, in this case a generalized RTC that fits Gemini requirements and one for the GNAO. The top level context diagram in Figure 2-1 is an example of a Template RTC using HEART. The context for the GNAO RTC is the same except there is no Tip/tilt Stage.

The RTC Command Handler accepts external commands (and internally distributes them to individual blocks as needed), and reports status back. The Hard Real-Time (HRT) Pipeline consists of all of the tasks that are required to be completed within one frame, or at most over a small number of frames (i.e., small fractions of a second). The Soft Real-Time (SRT) software consists of tasks that take much longer periods of times (e.g., several seconds or more). The Telemetry Storage System handles the different types of telemetry in this design.

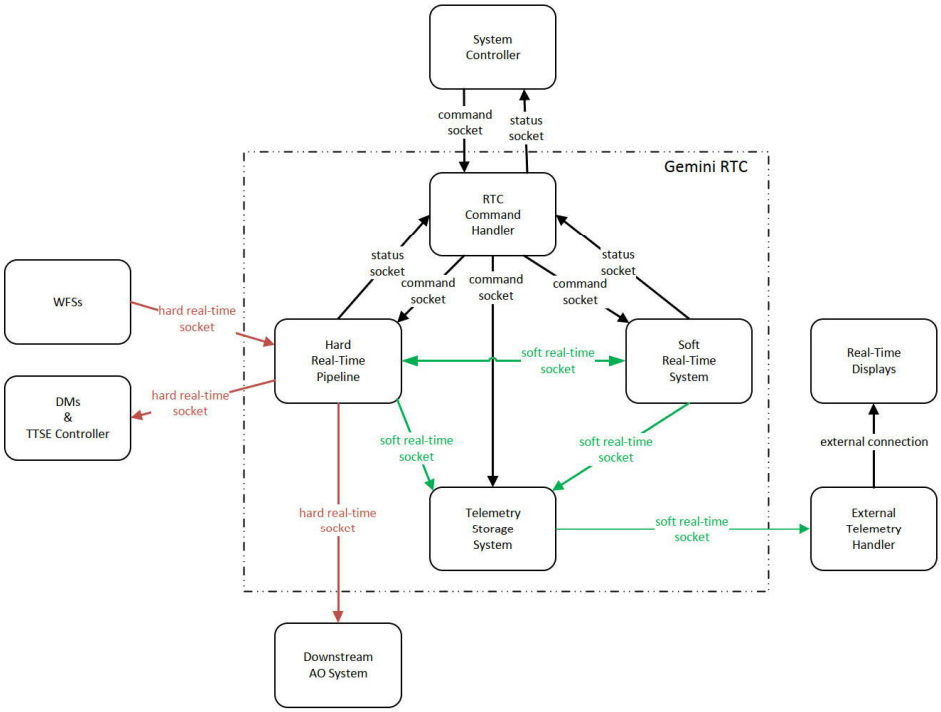


Figure 2-1 – GNAO NRTC Context Diagram

The Gemini Statement of Work (SOW) makes reference to a common code base and, in this design, it is making reference to HEART, except that HEART includes much more functionality than just a library of functions.

2.1.1 Template RTC

The Template RTC configures the HEART framework to create a working example of an MCAO RTC. All interfaces to the Template RTC are defined at the internal interface layer, excluding any custom interface/interpreter code blocks used to access physical hardware. These external interface blocks are instead configured to pass information received from external simulators that provide pixel stream sources and command sinks to the Template RTC. This means that the external simulators pass the data using the internal interface layer. In HEART, the external simulators can be configured to read input data from a file and write output data to a file, thereby simulating the operation of an AO system driven by the Template RTC.

This Template RTC provides a solid working example of how to use HEART to create an RTC and is a good example of a general RTC implementation. It further shows that the template can meet the GNAO requirements. There is an example template within the HEART library, as well as the Gemini specific Template RTC that can be used to create other RTCs (e.g., GNAO, GeMS).

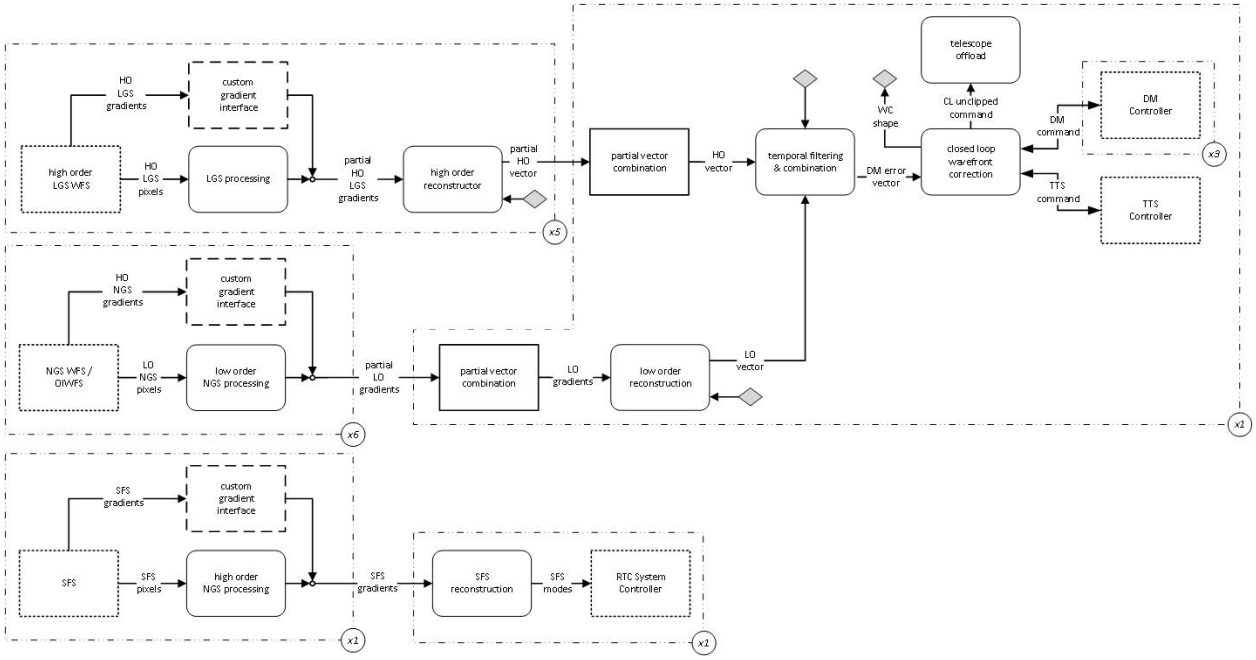


Figure 2-2 - Template RTC HRT block diagram built with HEART

The Template RTC HRT pipeline, as shown in Figure 2-2, is nominally modelled as a prototypical MCAO system. Five LGS WFS pixel streams are processed (LGS processing) and reconstructed in parallel. They are then summed and filtered (via a high order reconstructor) to produce a total High Order (HO) vector (partial vector combine). Three NGS WFS pixels streams, including those from On-Instrument Wavefront Sensors (OIWFSs), are also processed in parallel (via low order NGS processing) and the aggregated gradients are used to reconstruct a Low Order (LO) mode vector (i.e. low order reconstruction). The low order modes are filtered and projected into DM command space to be summed with the high order error vector (i.e. temporal filtering & combination). The net error vector is integrated and then decomposed into three virtual DMs, corresponding to different altitudes (i.e. closed loop wavefront correction). The ground-layer DM is further decomposed into DM commands and tip/tilt commands. These commands are filtered and resampled for telescope offloading (i.e. telescope offload). Slow focus sensor (SFS) pixels or gradients are processed (using high order NGS processing) to compute focus, and other modes, to be offloaded (via SFS reconstruction).

The overview of the Soft Real-Time (SRT) system, shown in Figure 2-3, includes:

- a command handler to accept commands and report status/state,
- optimization blocks which include slope, threshold, weights, frame rate and loop gain
- Reconstructor Parameter Generator (RPG) to calculate the control matrix
- and Calibration Tasks

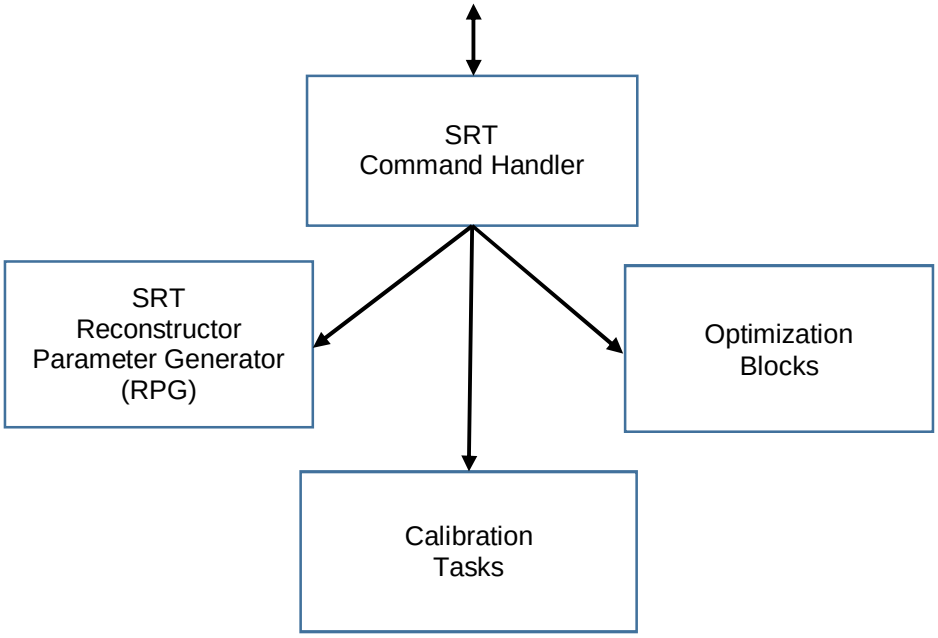


Figure 2-3 – Template RTC SRT Block Diagram built with HEART

The Command Handler interfaces with the Gemini RTC System Control software. It is the external interface to the RTC and handles commands being received and status reported out. This will be the same for both RTC's. The same is true for the Telemetry Storage System which handles the different types of telemetry in this design.

2.1.2 GNAO Template RTC

The GNAO Template RTC is very similar to the Template RTC, but will include the interface code blocks to GNAO components, such as WFSs and DMs. The GNAO Template will also include the specific configuration of the template for the system, including such details as: number of WFSs, number of sub-apertures per WFS, number of pixels per sub-aperture, number of DMs, number of actuators per DM, temporal filter parameters, control gains, reconstruction parameters, loop rate ranges, CPU core and memory allocations and ground layer tip/tilt decomposition parameters. Since the Template RTC is modelled after a generalized version of GNAO, this customization step will be straight-forward.

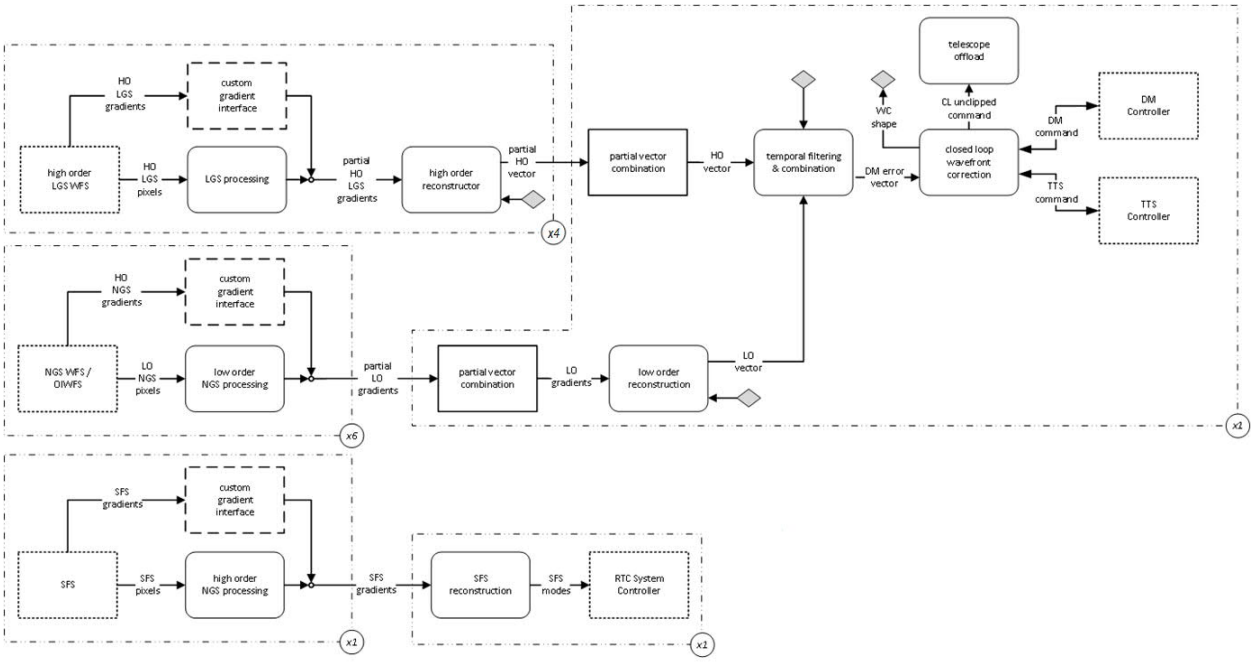


Figure 2-4 – GNAO Template RTC HRT block diagram built with HEART

The GNAO Template RTC HRT pipeline, as shown in Figure 2-4, is nominally modelled as a prototypical GLAO/SCAO system. Four LGS WFS pixel streams are processed (using LGS processing) and reconstructed in parallel then summed and filtered (via a high order reconstructor) to produce a total High Order (HO) vector (partial vector combine). One NGS WFS pixel stream, is processed (low order NGS processing) and the aggregated gradients are used to reconstruct a Low Order (LO) mode vector (low order reconstruction). The low order modes are filtered and projected into DM command space to be summed with the high order error vector (temporal filtering & combination). The net error vector is integrated and then decomposed into 1 virtual DM, corresponding to different altitudes (closed loop wavefront correction). The ground-layer DM is further decomposed into DM commands and tip/tilt commands. These commands are filtered and resampled for telescope offloading (telescope offload). Slow focus sensor (SFS) pixels or gradients are processed (high order NGS processing) to compute focus, and other modes, to be offloaded (SFS reconstruction).

The overview of the Soft Real-Time (SRT) system is the same as shown in the Template RTC section.

2.2 ROAD MAP

The focus of this document is to describe how the Template RTC and GNAO Template, including the customization of HEART required, will be used to develop a customized and functional RTC. This road map provides the reader guidance to the overall documentation to demonstrate that the customization of HEART to produce the Template RTC and GNAO Template are ready for build. This table contains the items that are listed in the SOW as deliverables for the Critical design and where to find that information in the documentation.

Table 2-1 - Roadmap

SOW Section 3.2.3	Description and Location

1. Software documentation: All the major software components shall be described at the level needed to code. The documentation shall be at a stage such that any senior software engineer can understand how the software system works.	
a. An overview of the software architecture.	Overview of the Template and GNAO Template RTC designs are in Section 2.1 and Section 3 where the changes since Preliminary Design are contained in Section 3. Both the Template and GNAO Template RTC designs are on the HEART design where more details on the design are found [RD1]
b. Description of all major software components	Section 10 contains the details on the Template RTC, Section 11 contains the details on the GNAO Template RTC. It includes block diagrams showing inputs and outputs, and points to further detail. The interfaces are also called out. Programming design constraints are shown in Section 6.1 and the performance budget is in Section 9.1.
c. Description of the instrument's status and commands.	Command structure, along with the commands and status provided by the HEART interface are covered in the External ICD [RD2]. Additional commands and status that is custom for the GNAO Template RTC can be found in [RD14].
d. Description of configuration file purpose and format	There is only 1 configuration file for each RTC that is used by all blocks. General details on the configuration file are found in the HEART design [RD1]. The Template RTC configuration file is in Section 10.1, and for the GNAO Template RTC configuration file is in Section 11.1
e. Description of development platform and tools	Section 6, and Development Process Document [RD7]
f. Description of processing design choices.	Section 2.4 covers the processing design choices made for the customizations required for Template and GNAO Template RTC
2. A final list of software configuration items identifying, for each configuration item, any existing code to be used, any existing code to be modified and used, and any code that is to be developed from scratch.	Same as #1b above, with the addition of Section 4 which contains information about existing/new/modified code.
3. A final dictionary of command and status items.	HEART External Interface Document contains the commands and status items [RD2] and also the GNAO Template ICD[RD14]
4. A build set of Internal System Interface Documents. All interfaces between internal and external hardware or software subsystems shall be finalized and documented.	HEART External Interface Document [RD2] and Internal Interface Document[RD15]. Since the GNAO instrument has not started the GNAO interfaces will be updated when possible.
5. A final list of software milestones, releases, and schedule points for Gemini	Section 6 discusses the development process, the milestones and schedule points are in Section 6.2.

collaborative testing with associated dates. Software release milestones shall be no less frequent than approximately once every four months.	
6. A final design of the computing hardware, networking, and any extra hardware needed to interface the computing system to external hardware.	List of hardware that will work for the GNAO Template RTC is discussed in Section 12
7. A final report on Technical Performance Metrics.	The key requirements are in Section 8 and in Section 9 the Technical performance metrics along with the timing budget
8. A draft Factory Acceptance Test (FAT) plan , Onsite Assembly and Installation (OAI) plan, and Onsite Acceptance Test (OAT) plan for each RTC System implementation.	FAT [RD10] OAI[RD11] OAT[RD12]
9. A complete phase plan for the Common Code Base Realization Phase	The Schedule[RD13] shows the HEART development interlaced with the custom parts required for the Template RTC and GNAO Template RTC
10. A complete phase plan for the Template RTC Implementation, Integration, Verification, and Validation phase	The Schedule[RD13] shows the build phase, along with the FAT testing.
11. A draft phase plan for the GNAO Implementation, Integration, Verification, and Validation phase	The Schedule[RD13] shows the build phase, along with the FAT testing. The build of the Template RTC will happen alongside the GNAO Template RTC.
12. A draft phase plan for the GeMS Implementation, Integration, Verification, and Validation phase.	Not applicable
The Critical Design shall be considered complete when (1) all of the tasks listed above are complete for the RTC System product, the Template RTC, the GNAO RTC, ~~and the GeMS RTC~~, and (2) the design is complete and demonstrated to meet all the stated requirements	

2.3 BLOCKS AND PIPELINES

Customization of HEART is straightforward due to its modular block architecture. Developers are free to customize the inputs and outputs of blocks, making it possible to insert/remove and change the order of calculations, as well as configuring different data dimensions and rates for each block. In HEART, the concrete assembly of the blocks into a system (including all of the input and output mappings for each block) is referred to as pipes or all together as a pipeline, see Section 9 for more details.

The pipeline has framework to:

- create/destroy,
- accept and fan out commands,
- combine status,
- change state of the blocks,
- provide a mechanism to trigger proceeding blocks based on data available using a semaphore system
- and reconfigure the blocks based on mode changes.

A pipeline will typically contain 1 or more blocks, usually grouping blocks together. HEART has sample pipelines, and both the Template and GNAO Template RTC will have similar pipelines. The major differences being the number of blocks running, which is based on the number of WFS's selected; there is also a difference in the number of outputs. The output and input blocks will be different. These differences are detailed in Section 9 and 10.

In all RTCs, the high and low order processing and reconstruction are in pipelines. The output from both of those pipelines are put into the Wavefront correction pipeline. Details of the pipelines used are in Section 4.

2.4 DESIGN CHOICES

There were design choices made during the design of the Template RTC and GNAO RTC. The primary design impacting decisions that were needed included software specific choices including:

- Whether triggering blocks could be done through semaphores, message queues or sockets
- Deciding on a non-realtime operating system vs a real time one
- Which were the best language options such as C, Java, Python, C++
- Option to make custom software and functions over using a block paradigm

Where the final software design choices were:

- Use of semaphores to trigger blocks is much more efficient and cleaner code. The messages queues would require more infrastructure (including more setup). Sockets were only slightly slower than semaphores and allow more flexibility but semaphores are simpler and are preferred when appropriate.
- Linux operating system with real-time extensions was chosen to get the increased real-time speed and superior deterministic timing.
- C was selected because it gives access to lower level functions that can help the compiler to optimize the code eliminating the overhead for a language interpreter. C was selected over C++ as the language itself is simpler, and does not hide (abstract away) what the CPU is doing.
- Blocks were selected to provide flexibility for the future to allow for re-configuration for the future customizations and AO systems.

Processing design choices included:

- Both Gemini and HAA worked on testing the tomographic reconstructor matrices computed using OOMAO and YAO to ensure that the tomographic reconstructor works.

2.5 INTERFACES

The Template RTC will have interfaces with the following simulated inputs:

- WFSs (5 LGS)
- DMs (3)
- TTS
- SCS (this is unique for Gemini)
- LGS Fast Steering Mirrors (FSM) or Jitter Mirror
- Gemini System Controller interface.

The GNAO Template RTC will have the following real interfaces:

- WFSs (4 LGS)
- DM (1)
- TTS
- SCS (this is unique for Gemini)
- Slow Focus Sensor (SFS) (unique for Gemini)
- LGS Fast Steering Mirrors (FSM) or Jitter Mirror
- Gemini System Controller interface.

An RTC, such as the GNAO RTC, is a subsystem with many interfaces, both internal and external. [RD1] and [RD2] provide standard default interface details and how to customize those interfaces for the specific GNAO RTC interfaces. Since the GNAO instrument hardware is unknown at this time, only the protocol is defined, not the details. The major interfaces for the GNAO Template are shown in Figure 2-5.

Command structure, such as the following, is covered in the External ICD [RD2].

- Definition of different command types
- Command properties
- Standard commands that are available on all pipelines and blocks
- Standard states
- Command state attributes
- Standard command status
- Notable events and alarms

There is an interface with the Gemini System Controller that includes:

- Commands from the System Controller to the RTC Command Handler
- Status out of the RTC:
    - state of the RTC,
    - LGS Focus Errors,
    - Focus and coma correction for the Secondary Control system,
    - NGS centering offsets for the NGS WFS
    - Field rotation offsets
    - Primary mirror figure error corrections

Details of all of these interfaces, including a command and status are covered in the External ICD [RD2]. Additional command and status that is specific to the GNAO Template RTC can be found in [RD13].

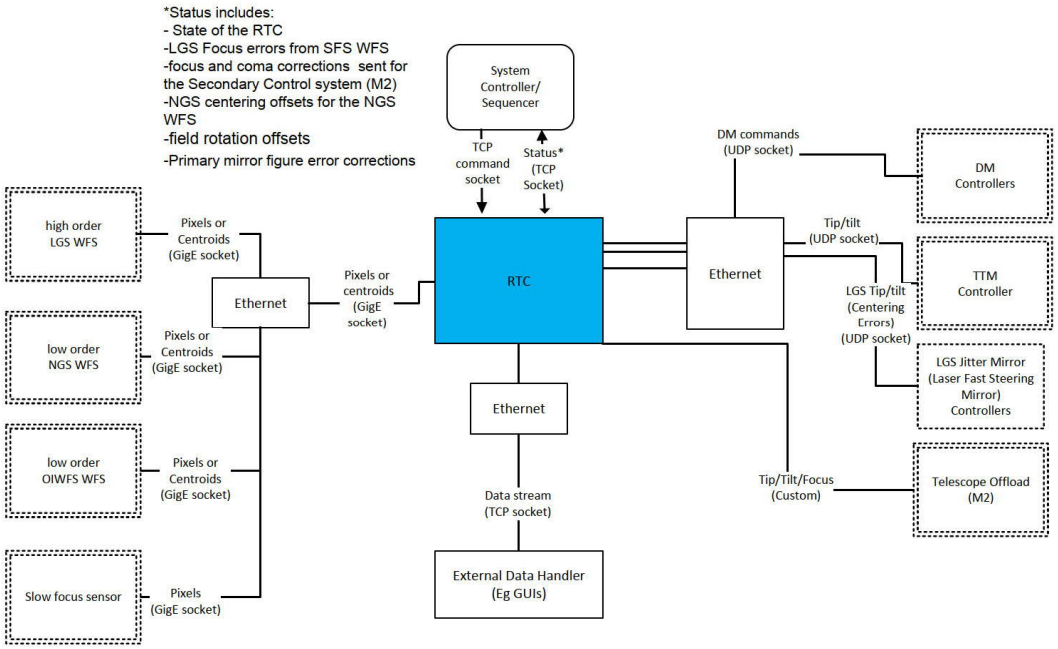


Figure 2-5 – GNAO Template Interfaces

In general the System Controller outward interfaces to the RTC are the following:

- commands the RTC to set up and control the AO loop
- status to and from the RTC
- report LGS Focus errors (also known as LGS TT) (nm RMS)
- focus and coma corrections destined for the Secondary Control system (M2)
- NGS centering errors (difference between the centroid and slope offset) for the NGS WFS controller
- field rotation offsets
- Primary mirror figure error corrections
- External Data Handler (eg. for a GUI) and an interface for a telemetry stream, which is provided to the configured external socket. The expectation is that that external data handler would provide data streams to any number of GUI's.

Data input interfaces to the RTC are physically through an Ethernet connection (will need to be updated when the GNAO instrument design progresses):

- Raw pixels or centroids from multiple LGS WFSs
- Raw pixels or centroids from multiple NGS/OIWFS WFS
- Raw pixels from SFS WFS

Outputs are sent from the RTC over an Ethernet connection:

- NGS Tip/tilt to the Tip/tilt stage electronics (TTSE) controller (filtered, and cutoff)
- Deformable mirror (DM) positions to the DM controllers (low, medium and high altitude)

- LGS centering errors to the Laser Fast Steering Mirror controller
- Secondary control mirror tip/tilt/focus

The interfaces shown above for the Template RTC, by definition, such that the system can be run without access to any external hardware. What this means is that the inputs are simulated data via software, not hardware. And the outputs also go to a software sink.

3. MODIFICATIONS DONE SINCE PRELIMINARY DESIGN REVIEW

The following details the changes to the design and to the documentation since the Preliminary Design review.

They include:

- The HEART Design document had most of the Template and GNAO references removed and added to this document. This change allows for a clearer description within a single document to describe this specific design.
- Additional details of the HEART design have been added.
- An Internal Interface document was created to detail the interactions between the blocks, definition of the pipelines, and block specific commands.
- The External Interface document was expanded to include more detail on commands, their structure, and status indicators. This includes a list of commands that are accepted by HEART.
- The GNAO instrument has not yet been assigned, but an initial draft of the commands and status that are specific for the GNAO is outlined.
- Additional design work on the SRT, in particular the Reconstructor Parameter Generator (RPG). Work with Gemini on the algorithms for the RPG were converged on. Also update to the POLC implementation as specified by Gemini.
- Design expanded for the Telemetry system.
- Clarification of the jitter and latency budget.
- Benchmarking of a typical Template system.
- Updating the drafts for the Factory Acceptance Test (FAT), Onsite Assembly and Installation (OAI) plan, and Onsite Acceptance Test (OAT) plan

Architectural changes made since PDR:

- The GNAO Template RTC changed from an MCAO system to LTAO/GLAO, but the Template RTC has not changed.
- Since Gemini has not provided a formal Algorithm Description Document for the “High Order Reconstructor”, it was agreed that the Soft Real Time System will compute this reconstructor by implementing the code currently used in OOMAO, the HAA end-to-end AO simulation software. This is described in Section 5.4.3.

4. DESCRIPTION OF ALL MAJOR SOFTWARE COMPONENTS

This section covers the major software components (or software configuration items) for both the Template RTC and GNAO RTC. This will also list any existing code to be used, any existing code to be modified and used, and any code that is to be developed.

The code is being kept in a Git-repository and is divided into 4 different repositories:

daoinsw is the HAA toolkit library that contains mature, thoroughly tested, code that has been utilized in many different projects. It provides common functionality such as debugging helpers, communication libraries, math functions, synchronization, etc. This existing library provides core reusable functionality for many different components and is written in C.

HEART is the HAA developed RTC repository. This contains the blocks and pipelines and templates to build finalized RTCs. This is currently being built and written in C.

TemplateRTC, which contains the customization required for the HEART blocks and the Template RTC. This includes the custom interface to the Gemini System Controller, and the MCAO template specific to the Template, an MCAO system that contains 5 LGS WFS and 3 DMs. This customization will be built based on the templates created for HEART. It also will be written in C.

GNAORTC, which contains the customization required for the HEART blocks and the Template RTC. This includes the custom interface to the Gemini System Controller, and the GLAO/SCAO template specific to the GNAO Template that contains 3 LGS WFS and 1 DM. This customization will be built based on the templates created for HEART. It also will be written in C, and this is dependent on the custom drivers required to interface with the RTC specific hardware.

4.1 Breakdown of a HEART Based RTC System

For both the Template and GNAO Template RTCs, there will be a main process that will contain the HRT pipelines. This main process is responsible for starting up the RTC main software components, as well as managing the control communication coming into the system and relaying the necessary command and information to the internal components.

To facilitate the full workflow for command and control of the RTC, the System Controller will communicate directly to the main process' external Command Handler. The Command Handler is the main communication point to interface with the RTC subsystems. Should the System Controller not be able to communicate directly with the Command Handler (i.e. it does not support the main protocol used by the Command Handler), a Custom Interface interpreter will seamlessly manage this communication. Messages sent to the Command Handler (either directly or indirectly through an interpreter), are then interpreted by the Command Handler and forwarded to the appropriate pipe as illustrated below.

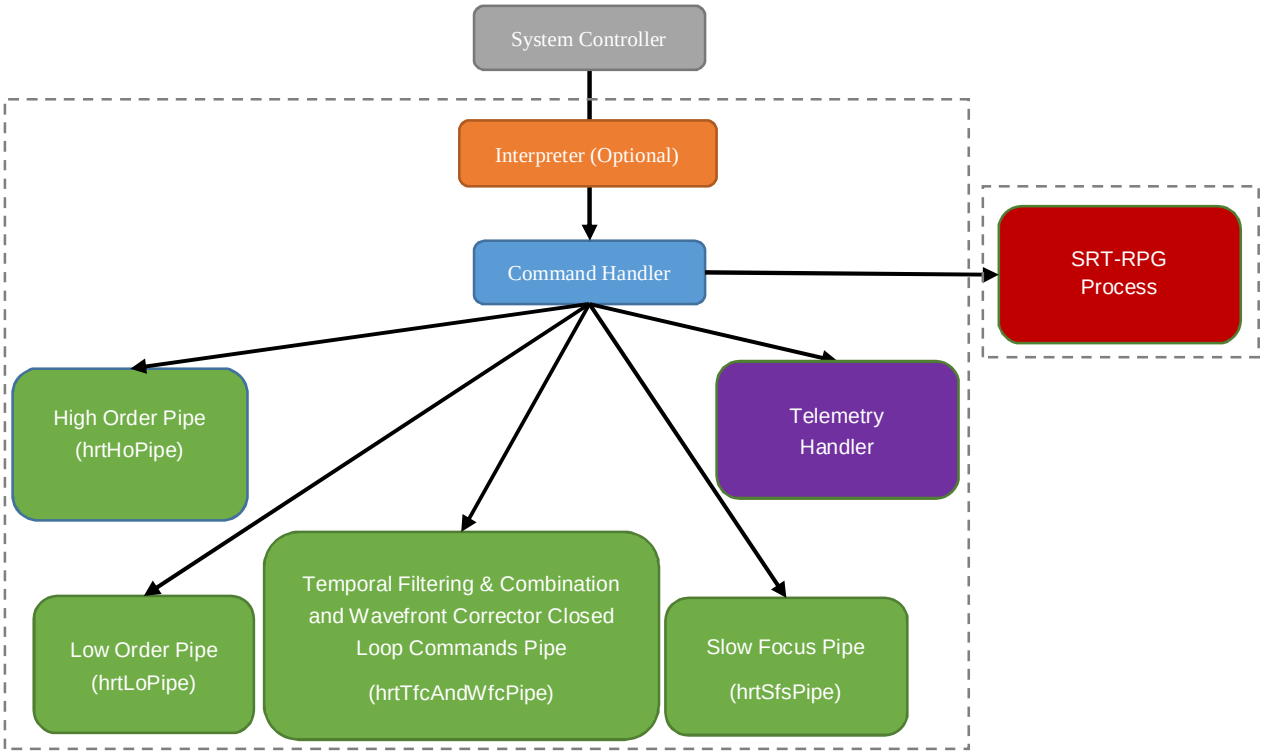


Figure 4-1 – HEART Based RTC Software Breakdown Diagram

Once receiving the specific internal commands, the individual pipelines will forward the necessary information to the appropriate block to command and configure its operation. The block will then operate as instructed by the system controller. Each pipeline contains blocks relevant to the pipeline functionality and is outlined in the table below.

Table 4-1 – Template HRT Pipeline Breakdown

HRT Pipeline	Block
High Order Pipe (hrtHoPipe)	LGS WFS input ($x5_{LGS}$)
	LGS Processing ($x5_{LGS}$)
	HO Reconstructor ($x5_{LGS}$)
	Vector Combine (x1)
Low Order Pipe (hrtLoPipe)	NGS WFS input ($x6_{NGS/OIWFS}$)
	NGS Processing ($x6_{NGS/OIWFS}$)
	Vector Combine (x1)
	LO Reconstructor (x1)
Slow Focus Pipe (hrtFigurePipe)	SFS input (x1)
	NGS Processing (x1)
	SFS Reconstructor (x1)
	SFS mode output ($x1_{System\ Controller}$)
Temporal Filtering & Combination and Wavefront Corrector Closed Loop Commands Pipe (hrtTfcAndWfcPipe)	Temporal Filtering & Combine (x1)
	Closed Loop WFC (x1)
	DM output ($x3_{DM}$)
	TTS output ($x1_{TTS}$)
	Telescope Offload (x1)

Should more extensive or fine-grained control be required, a software command client will be provided to access direct functionality such as engineering commands or status information specific to individual pipelines or blocks.

The main process of the RTC also starts threads for managing the telemetry data in circular buffers and write it to disk.

Additionally, to the main process of the RTC, there will be a separate process that will start and run the SRT-RPG. The SRT is responsible for the co-processing tasks such as generating appropriate parameters for the HRT tasks, optimizing these parameters as conditions change, performing calibration tasks, providing diagnostic data, etc. with most of which are used for optimization purposes. Mostly, it provides calculations, controls, and gathering of statistics of slower tasks not critical to the real-time execution of the RTC. The details of the SRT can be found in Section 5.4 as well as being described at length in Section 5 of the HEART Design Document [RD1].

5. RTC SOFTWARE COMPONENTS

Given the components of a HEART based RTC system, the Template RTC will consist of the following individual software components:

- A Command Handler which accepts external commands and internally distributes them to individual blocks as needed. This is common for all Template RTCs and the custom part of the design is discussed in Section 12.
- The Hard Real-Time (HRT) Pipeline consisting of all of the tasks that are required to be completed within one frame, or at most over a small number of frames (i.e., small fractions of a second) (Section 5.2).

- The Soft Real-Time (SRT) software consisting of tasks that take much longer periods of times (e.g., several seconds or more) (Section 5.4).
- The Telemetry Storage System which handles the 4 different types of telemetry.

The following sections will detail how the generalized heart blocks are used to create the Template RTC.

5.1 MODIFICATIONS REQUIRED FOR GEMINI

In order to meet the requirements of the Gemini RTC, a particular example pipeline will be custom built specifically for Gemini, called the Template RTC. While the Template will contain a custom example for Gemini, the blocks themselves and the framework connecting those blocks are built using the generic source code that is part of HEART. The most notable customization will come with blocks that have interfaces with hardware (e.g., receiving pixels from the various WFSs, or commands for WFCs), which will generally channel data through an Ethernet connection, but this may not always be the case. For example, from the RTC's point of view, by default WFS pixels are delivered via a socket; however, to support any existing or future WFS, DM or other external component, a developer need only develop a thin interface function that can be configured into the pipeline that connects the arbitrary external interface to the internal HEART data structures.

For Gemini, the main customizations required are the particular hardware interfaces. RTC shall directly send or receive:

- LGS WFS pixels or gradients to the RTC,
- NGS WFS and OIWFS pixel or gradients to the RTC,
- SFS pixel or gradients to the RTC,
- wavefront tip-tilt corrections to the Tip-Tilt Mirror Controller,
- calculated Deformable Mirrors shapes to the corresponding Deformable Mirror,
- commands to the LGS Jitter Mirror (Laser Fast Steering Mirror) Controller, and
- telescope offloading values to the System Control and Secondary Mirror Control System

Another customization that will be required is the interface between the Gemini System Controller and the RTC Command Handler. The Gemini System Controller sends commands and status and also receives status from the RTC System. Some of the status sent to the Gemini System controller is specific and will require some customization to get the correct orientation and units:

- WFS centering errors
- field rotation offsets
- send the Slow Focus errors (LGS focus errors) for the LGS Zoom Lens
- primary mirror figure corrections (filtered and down sampled)
- focus and coma corrections (filtered) for the Secondary Mirror Controller

It is worth mentioning that the HEART design supports a substantially more complex matrix of low-order wavefront sensor configuration and control paths than what is required by GNAO. In particular, HEART uses a concept of "Tier" assignments for the various low-order sensors to classify how they will be used (for example, whether they are only for tip/tilt, or tip/tilt/focus). While a full description of HEART Tiers is not required here, the general summary is that: Tier 0 is a moderate to high order NGS WFS for initial acquisition of the low order guide stars, Tier 1 is a low order NGS WFS that is capable of measuring focus used to stabilize the image before adding addition low order guide stars, Tier 2 is one or more low order NGS WFS that measures TT and along with Tier 1 correct all low order modes, Tier 3 is one or more low order NGS WFS that measures TT and optionally focus provided low order truth correction on slow time scales. It

suffices to say that all GNAO RTC low-order wavefront sensors are “Tier 2”, tip/tilt sensors in the HEART terminology (noting that focus is handled in a completely separate control path in GNAO). As discussed in section 9.7, the LO processing path can be configured with or without pseudo open-loop control. Thus, even though the Template RTC will support the existing Gemini System, it will be ready for any upcoming upgrade or new functionality in the future.

5.2 COMMAND HANDLER

The RTC Command Handler is the primary entry point into the RTC. It is responsible for performing much of the startup such as reading configuration, creating circular buffers, creating and starting pipelines, and the overall general initialization and management of the RTC. Once established, the command handler maintains its management responsibilities as it actively accepts external command, reports status, and monitors blocks.

For the Gemini system, the Command Handler has an interface that is specific to Gemini and acts as a conduit between the Gemini System Controller and the RTC. This is performed through the standard command handling mechanisms as well as through the customized Gemini interface. Through both of these interfaces, the Command Handler accepts commands, subscribes to status, and publishes status. The commands and status that are unique to Gemini are detailed in the GNAO Template ICD [RD13], with the standard commands detailed in the External ICD [RD2].

Further details on the design of the command handler can be found in Section 6 of the HEART Design Document [RD1].

5.3 HARD REAL-TIME PIPELINE

Figure 2-2 shows all of the blocks required to realize the Template RTC. The double lines around many of the blocks indicate that the block may be repeated in parallel (e.g. one block per WFS). This assumes that each instance of a block is executed on a single machine, however discreet blocks may be distributed across many machines (with some limitations due to the overhead of transmitting data between servers). Blocks with dotted lines indicate external hardware, primarily WFSs, DMs and TTSs.

A hierarchical architecture has been adopted which uses the following top-level blocks:

- LGS WFS Processing
- Low Order NGS WFS Processing
- High Order Reconstruction
- Low Order Reconstruction
- Slow Focus Sensor Reconstruction
- Partial Vector Combination
- Temporal Filtering & Combination
- Closed Loop Wavefront Corrector Control
- Telescope Offload

Rounded-cornered blocks mean that there is a corresponding sub-block that gives an expanded view of data flow. Note that some interfaces such as those to the soft real-time system or to the LGS jitter mirrors are only shown in sub-blocks to reduce clutter at the top level. In several of the sub-block diagrams there are blocks with dashed boarders. These blocks are areas where custom code is expected, primarily to handle custom interface details. Default example blocks will be made available to provide hooks for the custom code and those custom details are included below. The Template RTC pipeline supports LGS MCAO, GLAO, and LTAO correction modes.

Multiple High Order LGS WFSs stream pixels to corresponding LGS Processing blocks or gradients to a corresponding High Order Reconstruction block.

The High Order Reconstruction blocks receive gradients streams from corresponding LGSs to produce HO vectors. If there is more than one High Order Reconstruction block (i.e. the reconstruction process has been parallelized based typically on the WFSs), then these partial HO vectors are aggregated and combined by a Partial Vector Combination block to produce the HO vector. The HO vector is then sent to the Temporal Filtering & Combination block.

Low Order NGS WFSs stream pixels to the corresponding Low Order NGS Processing blocks. These WFSs can be moderately high order WFS (tens of sub apertures), tip/tilt sensors, On-Instrument WFS (OIWFSs), or any combination thereof. The Low Order NGS Processing blocks then send LO gradients (or modes) to the Low Order Reconstruction block. Typically, there is a single Low Order Reconstruction block that collects all LO gradients and modes together. It is expected that this may not necessarily be a required restriction in the framework. The Low Order Reconstruction block produces a LO vector that is sent to the Temporal Filtering & Combination block.

A SFS WFS can also be used, which is processed by High order NGS Processing. The gradients are then sent to the SFS Reconstruction block where they are converted to low order modes which are, in turn, sent to the RTC System Controller.

The Temporal Filtering & Combination block collects HO and LO vectors into a single DM error vector. Typically for a MCAO or SCAO system, there would be a single Temporal Filtering & Combination block. The DM error vector is then sent to a Closed Loop Wavefront Correction block (for an MCAO or SCAO system).

For an MCAO or SCAO system, a Closed Loop Wavefront Correction block integrates the DM error vector and decomposes it into the appropriate wavefront corrector elements, such as a Tip/Tilt Stages (TTS) and one or more DM. These wavefront corrector elements are then sent commands to update their shape/position.

The Closed Loop Wavefront Correction block sends copies of the WC commands to the Telescope Offload block. This block aggregates all the WC commands and offloads persistent shapes to the Telescope Control System (TCS) to reposition/reshape any of the mirrors within the telescope.

The Slow Focus Sensor (SFS) path is custom for Gemini and is constructed using the same blocks as the LO path, except the output modes are redirected to the system controller via custom code instead of going to the Temporal Filtering & Combination block.

The mapping of block processing tasks to computer hardware is specified in a configuration file. In order to make the mapping from tasks to hardware more clear, a single hardware configuration file will contain all of the mappings for a given AO system.

The following section will detail how the pipeline is constructed to create the Template RTC.

5.3.1 Constructing the Pipeline for the Template RTC

As described in Section 2.6 (hrtBlocks and hrtPipes) of the HEART design document [RD1], pipelines are constructed from HEART blocks (reusable components that perform a particular calculation), with information flowing between blocks facilitated by circular buffers (CB). Blocks are configured when instantiated using information provided in configuration files. At run-time, blocks are sent commands over UDP sockets to manage their state (e.g., to start/pause/stop), and to modify their configuration as needed. Triggering relationships can also be established between blocks so that an upstream block can notify one or more downstream blocks that new data is available in CBs for processing. Circular buffers store the recent history of data frames produced by a writer block, and one or more downstream reader blocks can access any of that history, including jumping to the most recent frame. Both the CPU cores upon which block worker

threads execute, and the memory nodes where CBs and other data are allocated are all configurable to optimize performance.

An additional abstraction called a hrtPipe manages the creation and operation of collections of blocks that typically operate in unison (e.g., to facilitate the “opening and closing of loops”). Each hrtPipe has a particular creation function that uses inputs from configuration variables to instantiate all of the blocks within the hrtPipe, and to establish all of the connections between the blocks (creation of CBs, providing them as inputs/outputs to the relevant blocks, and to define the triggering relationships).

At a high level the Template RTC is a grouping of elements of the block diagram (Figure 2-2) into the following hrtPipes (that will generally be operated in unison):

- hrtHoPipe: all of the HO LGS WFS processing up to the output of HO vectors
- hrtLoPipe: all of the LO NGS processing up to the output of LO vectors
- hrtTfcAndWfcPipe: Temporal Filtering and Combination of outputs from hrtHoPipe and hrtLoPipe, and generates Wavefront Corrector Commands (for DM and TTS, and telescope offload)
- hrtSfsPipe: Slow Focus Sensor processing up to the output of SFS modes

In order to orchestrate the activities of the blocks and hrtPipes, the Command Handler is responsible for receiving external commands (i.e., from the System Controller), and fanning them out into individual commands for each block/hrtPipe, and also providing responses.

For a list of all configuration items used across the Template RTC, please refer to Section 3 of the HEART Internal Interface Document [RD14]

Specific configuration items, and details about the different hrtPipes and hrtBlocks are provided in the following subsections.

5.3.1.1 hrtHoPipe

This hrtPipe consists of the following blocks:

- N x **hrtWfsInputBlock** (LGS pixel reception) that provides the interface to receive pixels from the WFS
- N x **hrtPixelProcBlock** (LGS pixel processing) that processes pixels into gradients.
- N x **hrtMvmBlock** (HO reconstruction) that produces partial HO vectors from the gradients produced by the preceding hrtPixelProcBlocks using matrix vector multiplication.
- 1 x **hrtVecCombBlock** (Partial Vector Combination) that produces a single HO vector from the five partial vectors.

Each LGS WFS input block receives pixels from a single LGS WFS sensor. The interface to the WFS is established by providing a **wfsReader()** function pointer to **hrtWfsInputBlock_create()** (the constructor for the block). In the case of the template, an example of this function is provided to receive simulated pixel streams over a UDP socket. The hrtWfsInputBlock will then write the raw pixel values from the entire sensor into circular buffers cbLgsRawPixelsX (one for each LGS WFS) as they arrive.

The pixel processing blocks continuously monitor the raw pixel CB input streams, and calibrate the data using the supplied bias, dark, flat field and sky background frames in the CBs cbLgsFlatX, cbLgsDarkX, cgLgsBiasX, and cgLgsSkyX. Each is initialized via configuration parameters but can be updated dynamically during runtime by adding new values to the CBs, and then issuing the PIXEL_PROC_UPDATE_CAL command. This will cause the CBs to be checked for new values and updates them if needed. The resulting calibrated pixel values are written to circular

buffers cbLgsPixelsX. Configuration file entries also describe the layout of subapertures in the image. In this way, the pixel processing block can keep track of when the pixels for each subaperture are complete. At this point, the dot product with the gradient coefficients, stored in the cbGradCoeff CBs, are triggered. Again, the per-subaperture gradients are written to CBs cbLgsGrad as the gradients are calculated, rather than waiting until all the pixels are calibrated. Also custom is the **lgsFsmDriver()** handler function for the LGS centering errors for the Laser Fast Steering mirrors (i.e. jitter mirrors). As with the other custom handlers, sample functions are provided with the template that send simulated values over UDP sockets.

The MVM block performs a streamed MVM of the control matrix and the gradients are fed into cbLgsGradX, and writes the resulting partial HO vectors to circular buffers cbPartialHoX. As it is doing this, it also projects WC shapes into gradient space (i.e. the cbWcShape is created by the hrtTfcAndWfcPipe, and is provided as a reference using the MVM_WC_CB command) and adds them to the HO vectors prior to the MVM (in the case that POL feedback is being used). The control matrices are provided via cbCmHoX. The control matrices are updated dynamically by adding new values to the CBs and sending the MVM_UPDATE_MATRIX command to swap them in.

The list of the CBs created, the corresponding special command blocks, and the configuration items associated with the hrtHoPipe can be be found in Section 3 of [RD14].

NOTE: The Template RTC has been created to be flexible and expandable for any RTC system. For the GNAO RTC, the N^{LGS} supports up to a maximum of 5 to meet the requirements of GNAO.

5.3.1.2 hrtLoPipe

This hrtPipe consists of the following blocks, noting that there are a variable number of LO WFS sensors, denoted N:

- N x **hrtWfsInputBlock** (NGS pixel reception) that provides the interface to receive pixels from the WFS.
- N x **hrtPixelProcBlock** (NGS processing) that processes pixels into gradients.
- One **hrtVecCombBlock** (Partial Vector Combination) that produces a single LO gradient vector from the individual gradient vectors.
- One **hrtMvmBlock** (LO reconstruction) that produces LO vectors from the gradients produced by the preceding hrtVecCombBlock using matrix vector multiplication.

The LO NGS WFS processing is very similar to the HO LGS WFS processing, with three main differences. First, a different **wfsReader()** custom handler is required for the hrtWfsInputBlock. Second, the output of pixel processing is fed directly into a hrtVecCombBlock which concatenates all of the gradients into a single gradient vector. Next, the WC shape vector is projected into gradient space is added to the concatenated gradient vector if POL feedback is being used. Finally, there is a single hrtMvmBlock used to produce the LO vector from these gradients.

Another difference is that each block operates in the non-streaming mode (i.e., they await all of the data for a frame to arrive before processing begins). This behavior results in less CPU overhead spent on busy loops that await pixels as does the HO case (required to minimize lag in that case). It is feasible here because a relatively small number of pixels and gradients are produced in the LO path. Furthermore, for a detector which sends a small number of pixels or gradients, only a single datagram is required to transfer all of the data to the block so there is no latency reduction from using a streaming approach.

The list of the CBs created, the corresponding special command blocks, and the configuration items associated with the hrtLoPipe can be found in Section 3 of [RD14].

NOTE: The Template RTC has been created to have the flexibility and expandability for any RTC system. For the GNAO RTC, the N^{NGS} supports up to a maximum of 6 to meet the requirements of GNAO.

5.3.1.3 hrtTfcAndWfcPipe

This hrtPipe combines the LO and HO paths, and also generates demands and processes feedback from wavefront correctors. It consists of the following blocks:

- One **hrtTfcBlock** (Temporal Filtering and Combination) that merges the LO and HO vectors, includes WFC feedback, and produces the DM Error vector.
- One **hrtClWfcBlock** (Closed Loop Wavefront Correction) that derives DM and TTS commands from the DM Error Vector, and also generates WFC shape vector feedback for the hrtTfcBlock and also the hrtHoPipe and hrtLoPipe.
- One **hrtTelOffloadBlock** (Telescope Offload) that projects unclipped DM commands into telescope modes which are then passed through a temporal filter. The filtered tip and tilt modes are sent directly to the secondary control system at a reduced rate. This also includes the remaining filtered & down samples modes that are sent to the Command Handler where they are forwarded on to the System Controller.

The hrtTfcBlock performs post-processing on both the HO and LO vectors prior to combination in order to remove POL vectors (if POL feedback is being used). The WC shape needed for this feedback, cbWcShape, is created by the hrtClWfcBlock. Filters are then applied separately to the HO and LO vectors before they are added. This summed error vector is written to cbDmErr.

Next, the hrtClWfcBlock derives commands for the different wavefront correctors from cbDmErr. This derivation includes the results of Loop Gain Optimization. This block also writes both clipped and unclipped commands to the CBs cbClClipped and cbClUnclipped and, as mentioned above, cbWcShape. The interfaces to the various WC is established by providing the **dmDriver()** and the **ttDriver()** custom handler function pointers to **hrtClWfcBlock_create()**. These functions are called every time there is a new frame of commands to send, and a status response returned by each function includes sensed shape and clipping information. In the case of the template, an example of this function is provided to send simulated command streams over a UDP socket.

In parallel, the hrtTelOffloadBlock reads each frame of unclipped commands from cbClClipped and performs an MVM to produce telescope offload modes. The **hrtTelOffloadBlock_create()** function accepts a **telOffloadDriver()** custom handler function to send tip/tilt commands directly to the secondary control system. The remaining modes are written to cbTelModes and are sent to the Command Handler (to a socket at the IP address and port specified in the CH_IN_STREAM_IP configuration item) which is responsible for sending the offloads on to the System Controller.

The list of the CBs created, the corresponding special command blocks, and the configuration items associated with the hrtTfcAndWfcPipe can be be found in Section 3 of [RD14].

5.3.1.4 hrtSfsPipe

This hrtPipe reads data from the Slow Focus Sensor (SFS) and uses it to reconstruct LO modes that are sent to the System Controller. This is then used by the System controller to adjust the focus of LGS lasers. The hrtSfsPipe consists of the following blocks:

- One **hrtWfsInputBlock** (SFS pixel reception) that provides the interface to receive pixels from the WFS.
- One **hrtPixelProcBlock** (SFS pixel processing) that processes pixels into SFS gradients.

- One **hrtMvmBlock** (SFS reconstruction) that produces low order modes from the SFS gradients.
- One **hrtStreamBlock** (SFS mode streaming) that streams the low order modes to the Command Handler where they are forwarded on to the System Controller.

The SFS processing is similar to LO NGS WFS processing, though it requires its own **wfsReader()** custom handler, and is simpler due to the fact that there is only a single SFS with no feedback to consider. The output of the MVM is a vector of low-order modes (cbSfsModes).

The hrtStreamBlock is configured to send the contents of the cbSfsModes to the Command Handler which is responsible for the interface to the System Controller. This is done via socket to the IP address and port specified in the CH_IN_STREAM_IP configuration item.

The list of CB created as part of the hrtTfcAndWfcPipe can be found in Section 3.3 with the corresponding special command block found in Section 3.4 of [RD14]

The list of the CBs created, the corresponding special command blocks, and the configuration items associated with the hrtSfsPipe can be found in Section 3 of [RD14].

Extensive details regarding the Hard Real-Time System can be found in Section 4 of the HEART Design Document [RD1].

5.4 SOFT REAL-TIME SYSTEM

The RTC will execute co-processing tasks that run over time scales much larger than a millisecond frame. These tasks, referred to as soft real-time (SRT) include generating appropriate parameters for the HRT tasks (such as the control matrix), optimizing these parameters as conditions change, performing calibration tasks and providing diagnostic data - most of which are used for optimization purposes.

All SRT tasks required for the Template RTC and the GNAO Template RTC are part of the HEART design and are described in [RD1]. The following sections provide additional details on specific configurations of the generalized HEART SRT for Gemini.

5.4.1 SRT Rates

The rates required by both the Template RTC and the GNAO Template are detailed in Table 5-1.

Table 5-1 - SRT Block rates

Component	Description
NGS and LGS Slope calculations optimization	The RTC System shall be capable of re-optimizing the NGS or LGS slope calculation parameters at least every 10 Seconds.
NGS and LGS reconstruction	The RTC System shall be capable of re-optimizing the NGS and LGS reconstructor parameters at least every 10 Seconds.
LO and HO Loop Gain	The RTC System shall be capable of re-optimizing the low order and high order loop gain parameters at least every 10 Seconds.

5.4.2 SRT Sizes

The dimensions of the MCAO configuration supported by the Template RTC are given in Section 11.2. Since the GNAO configuration (given in Section 11.3) is smaller, the SRT from the Template RTC will also directly support the GNAO configuration.

From the point of view of the SRT, supporting the GNAO configuration is given in Section 11.3.

The relevant sizes are given in Table 5-2. Here, the number of actuaors and number of slopes are obtained from the OOMAO configuration of the systems, and might differ slightly from the numbers provided elsewhere in the document.

Table 5-2 - Physical parameters for Template and GNAO RTCs

Parameter	**Template RTC**	**GNAO Template RTC**
Number of LGS WFSs	5 LGS WFS, 20x20 sub-apertures, pitch p=0.396m	4 LGS WFS, 20x20 sub-apertures, pitch p=0.396m
Number of LGS slopes	N_{sl} ~3,040	N_{sl}=2,432
Maximum number of layers	n_l=7	n_l=7
Maximum altitude of the layers	H_{max}=20km	H_{max}=20km
Maximum FOV	FOV=85”	FOV=85”
Telescope diameter	D=7.9m	D=7.9m
Number of DMs	3 (D_{DM}=21, 19, 19)	1 (D_{DM}=21)
Total number of actuators	N_{act}= 1,035	N_{act}= 393

5.4.3 RPG Computation of the MV Reconstructor

The most computationally demanding SRT task is the calculation of the control matrix (REQ-8.2.2.25-27), which needs to be updated every 10 seconds (REQ-8.3.3.6) in order to account for changes in conditions. The Template RTC will implement a zonal minimum-variance tomographic reconstructor.

For GNAO the HO and LO reconstructors are recomputed every ten seconds. The update accounts for changes in geometry and observing conditions (seeing and WFS SNR). The “split tomography” approach is used where the HO and LO reconstructors are computed separately.

For the minimum variance (MV) reconstructors, the computational parameters are chosen according to Table 5-3.

Table 5-3 - Parameter choices for computation of the MV reconstructor

Parameter	**Template RTC**	**GNAO Template RTC**
Maximum number of layers	n_l=7	n_l=7
Maximum altitude of the layers	H_{max}=20km	H_{max}=20km
Grid sampling	p/2=0.198m	p/2=0.198m
Number of evaluation directions	N_d=49	N_d=49

The size of the meta-pupil on the highest layer is:

$$D_{meta}=D+H_{max}*FOV=7.9+20000*85/3600*pi/180=16.16 \text{ m}$$

With a sampling of p/2=0.198 meter for each layer and a 7 layer atmosphere with altitudes [0, 0.5, 1, 2, 6, 12, 20] km the sampling of each layer is [42, 43, 44, 46, 54, 66, 83] pixels, giving [1764, 1849, 1936, 2116, 2916, 4356, 6889] elements in each layer

The total number of points for all the layers is: N_l= 21,826.

Total number of points to sample the pupil plane: 21 pixels in diameter = N_p= 352 elements.

Table 5-4 provides the size of the different matrices involved in computing the MV reconstructor:

Table 5-4 - Sizes of the matrices involved in the computation of the MV Reconstructor

Matrix	Template RTC	GNAO Template RTC
G_x	N_{sl} x N_l = 3,040 x 21,826	N_{sl} x N_l = 2,432 x 21,826
C_{xx}	N_l*N_l= 21,826 x 21,826	N_l*N_l=21,826 x 21,826
C_{nn}	N_{sl} x N_{sl} = 3,040 x 3,040	N_{sl} x N_{sl} = 2,432 x 2,432
R	N_l x N_{sl} = 21,826 x 3,040	N_l* N_{sl}=21,826 x 2,432
H_x	(N_d*N_p) x N_l = 17,248 x 21,826	(N_d*N_p) x N_l= 17,248 x 21,826
H_a	(N_d*N_p) x N_{act} = 17,248 x 1,035	(N_d*N_p) x N_{act}= 17,248 x 393
W	(N_d*N_p) x (N_d*N_p) = 17,248 x 17,248	(N_d*N_p) x (N_d*N_p) = 17,248 x 17,248
P	N_l x N_{act} = 21,826 x 1,035	N_l x N_{act} = 21,826 x 393
C	N_{act} x N_{sl} = 1,035 x 3,040	N_{act} x N_{sl} = 393 x 2,432

As discussed in section 5.3.1.1.4 of the HEART Design Document, the computation of the MV-Reconstructor is dominated by solving N_{act}=1,035 linear systems of N_l=21,826 equations. This takes about 24 second on a 2018 Mac laptop with six 2.9 GHz Intel i9 cores and is expected to run in well under 10s on the final RTC hardware.

The noise covariance matrix for both the HO and LO reconstructors is computed using the "PSD analysis of the X and Y slope time series" method (see Section 4.3. of the HEART Design Document). The temporal PSD of each slope is computed by the HRT System. The noise variance is computed as the median value of the last N elements of the PSF, where N is a configurable number.

The SRT System will also compute the interaction matrix for POL operations as per Section 5.3.3 of the HEART Design Document [RD1].

5.4.4 Implementation

Since Gemini has not provided a formal Algorithm Description Document for the "High Order Reconstructor", it was agreed that the Soft Real Time System will compute this reconstructor by implementing the code currently used in OOMAO, the HAA end-to-end AO simulation software. The OOMAO section computing the reconstructor has been extracted and made into a stand-alone Matlab program called: matrixComputation.m. This program computes the matrices G_x, H_x, H_a and C_{xx}^{-1} based on the system configuration. The Python code described in Section 5.3.1.1.4 of the HEART Design Document is then used to compute the Tomographic Reconstructor based on these matrices. The Matlab program will be translated into Python and implemented as part of the Soft Real Time System.

Since Gemini has not provided a formal Algorithm Description Document for the "Low Order Reconstructor", it was agreed that the Soft Real Time System will compute this reconstructor by implementing a Python program called ngsreconstructor.py provided by Marcos Van Dam. This program implements the equations from section 4.3 of the HEART Design Document.

5.4.5 Impact of change in frame rate

When the System Controller changes the loop rate, the RTC continues to use the same reconstruction matrices and the same loop gains. It is the responsibility of the System Controller

to adjust the loop gains before the loop rate is changed in order to make sure that the loop rate change will not make the system unstable. It is also the responsibility of the System Controller to request the computation of a new reconstructor that takes into account the changes in SNR on the WFS measurements due to the rate change. Before doing this though, the System Controller should wait until the RTC has time to accumulate fresh statistics.

When the System Controller changes the loop gains, or the temporal filters, the System Controller should first disable automatic gain optimization. Not doing it might cause the Soft Real Time System to override the change at the next optimization cycle, and might induce instabilities.

5.5 TELEMETRY STORAGE SYSTEM

The telemetry storage system for Gemini is the main software component to manage the storage and interfaces into the telemetry data. Internally, the Telemetry Storage System interacts with the Command Handler, Hard Real-Time, and Soft Real-Time systems to collect, store and share the telemetry data for the entire RTC system. It does this through its management of storing the telemetry to disk while providing interfaces that internal and external systems can use to access the data and retrieve state, status, and operational values. These systems can be either to external connections through an External Telemetry Handler or to Downstream AO systems connected to the HRT subsystem.

There are four primary types of telemetry data that will be provided as part of the Template RTC:

- State/Status Telemetry: provide System Controller with state, status, and other offload values. It is a socket stream over external network. It is non/soft real-time.
- Stored Telemetry: for engineering purposes, locally written to disk mounted by the System Controller and is non real-time
- External Streamed Telemetry: for external displays or other applications, decimated socket stream over external network and is soft real-time
- Downstream AO Telemetry: for downstream AO correction (e.g. GIRMOS), full-rate socket stream over a dedicated link, and is hard real-time

For more detailed information on the Telemetry Storage System and its design as part of the HEART design, refer to Section 7 or the HEART Design Document [RD1].

For the Template and GNAO Template RTC the circular buffers shown in are an example of the selected Stored Telemetry.

Table 5-5- Example Stored Telemetry Circular Buffers

Name of circular buffer (see circular buffer declaraction in [RD14]	Description	Format
cbLgsPixels[1..N]	Calibrated LGS WFS pixels	[NUM_HO_PIXELS]
cbLgsGradCoeff[X|Y][1..N]	LGS WFS gradient coefficients	[2 * HO_WFS_SIZE]
cbLgsGradThreshold[1..N]	LGS WFS Gradient Threshold	[HO_WFS_SA]
cbLgsGradWeight[1..N]	LGS WFS Gradient Weight Map	[HO_WFS_SIZE]
cbLgsGrad[X|Y][1..N]	LGS WFS gradient	[HO_WFS_GRADS]

cbLgsSubApMask[1..N]	LGS WFS sub-aperture Mask	[HO_WFS_SA]
cbLgsSubApCoord[X|Y] [1..N]	LGS Sub-aperture X,Y coordinate map	[2 * HO_WFS_SA]
cbLgsSubApFlux[1..N]	LGS WFS sub-aperature flux	[HO_WFS_SA]
cbLgsFlux[1..N]	LGS WFS total flux	[1]
cbLgsTtErr	LGS TT err for the FSM	[2]
cbCmHo[1..N]	HO control matrices	[DM_SIZE] x [HO_WFS_GRADS]
cbImHo[1..N]	HO interaction matrices	[HO_WFS_GRADS] x [DM_SIZE]
cbPartialPol[1..N]	Partial HO POL gradient vectors	[HO_WFS_GRADS]
cbPartialHo[1..N]	Partial HO vectors	[DM_SIZE]
cbHo	Combined HO vector	[DM_SIZE]
cbHoLoopGain	High order loop gain	[1]
cbLoLoopGain	Low order loop gain	[1]
cbNgsGradCoeff[X|Y][1..N]	NGS WFS gradient coefficients	[2 * LO_WFS_SIZE]
cbNgsGradThreshold[1..N]	NGS WFS Gradient Threshold	[1]
cbNgsGradWeight[1..N]	NGS WFS Gradient Weight Map	[LO_WFS_SIZE]
cbNgsGrad[X|Y][1..N]	NGS WFS gradient	[LO_WFS_GRADS]
cbNgsFlux[1..N]	NGS WFS total flux	[1]
cbCmLo	LO control matrices	[LO_NUM_MODES] x [LO_WFS_COUNT x LO_WFS_GRADS] x [2^ LO_WFS_COUNT - 1]
cbImLo	LO interaction matrix	[LO_WFS_COUNT x LO_WFS_GRADS] x [LO_NUM_MODES]
cbGradLo	Concatenated LO gradient vector	[LO_WFS_COUNT x LO_WFS_GRADS]
cbPOLLo	LO Pseudo Open Loop vector	[LO_WFS_COUNT x LO_WFS_GRADS]
cbLo	Combined LO output vector	[LO_NUM_MODES]
cbDmErr	Virtual DM error vector	[DM_SIZE]
cbWcShape	WC shape for POL feedback	[DM_SIZE]
cbClUnclipped	CL unclipped commands	[DM_SIZE]
cbClClipped	CL clipped commands	[DM_SIZE]
cbOlClipped	OL clipped commands	[DM_SIZE]
cbDm	DM commands	[DM_SIZE]

cbTt	TTS commands	[2]
cbFigDmVector	Figure DM Vector	[FIG_DM_SIZE]
cbCmTel	Telescope offload reconstructor	[TEL_NUM_MODES]
cbTelModes	Telescope offload modes	[TEL_NUM_MODES]
cbHoPSD	High-order residual PSD estimate	[srtNumPsdPts]
cbLoPSD	Low-order residual PSD estimate	[srtNumPsdPts]
cbHoGradPSD	High-order gradient PSD	[HO_WFS_GRADS]
cbLoGradPSD	Low-order gradient PSD	[LO_WFS_GRADS]

5.6 TESTING SOFTWARE

There will be an external client tester, to simulate external components, such as WFSs and DMs. The WFS simulators will provide pixel inputs to the RTC, while the DM simulators will act as sinks to the outputs of the RTC. This is described in more detail in the GNAO RTC FAT Plan [RD5].

6. AGILE DEVELOPMENT, TRACKING, MILESTONES

6.1 AGILE DEVELOPMENT OF THE TEMPLATE AND GNAO

The Waterfall Model methodology has a linear life cycle where components are completed in order and the next component is not developed or tested until the previous step is completed. Waterfall is a traditional structured software development methodology that requires all components to be designed in detail upfront with little facilities for adaptability or allowance for change as development begins.

Unlike the waterfall methodologies, Agile development has a continuous iteration of development and testing occurring at the same time. This process allows more communication between the development team and the customer, Gemini. Because of the nature of iterative development it means that testing is built at the same time as the code which assures the quality of the development. And because of the iterations the development team and client know exactly what is complete and what is not, it greatly reduces the risk of the development process. Furthermore, because Agile is iterative, should changes in development or design be encountered during the development process, adaptations are easily integrated into the workflow achieving a better final design, implementation, and overall more desired end solution.

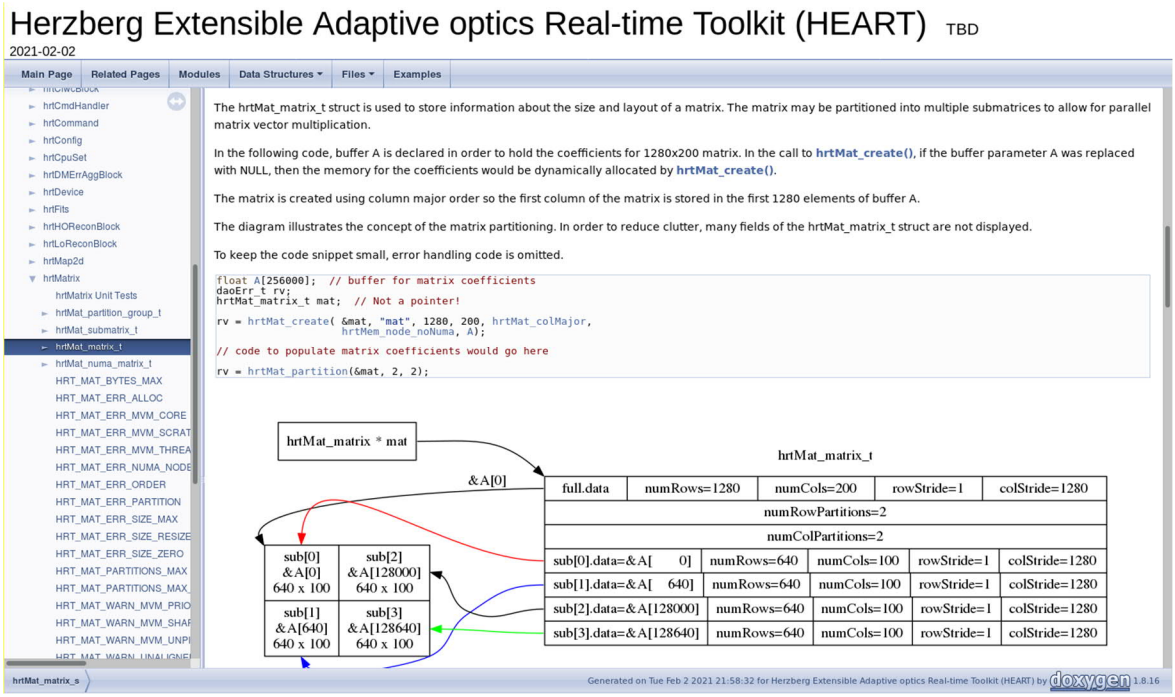
We are using an agile development approach for this project. The code is mainly written in C largely because of the frequency that the pixels need processing and calculations are performed to produce an output that will provide corrections to the wave front at least 1 kHz. Some of the slow-real time functions such as the Reconstructor Parameter Generator will take advantage of Python.

6.1.1 Documentation

This project is using built-in documentation created through Doxygen. This means that documentation is embedded in the code and built into a full suite of HTML files. An example from the documentation output is shown in Figure 6-1. The favorable part of using Doxygen is that it automatically generates links within the documentation to make it fairly easy to navigate as well

as the same documentation content is also embedded in the code. The embedded documentation makes it easily available to developers to understand implementation choices and update it as changes are made.

Figure 6-1 - Sample of documentation generated from the Circular buffer code



6.1.2 Management

In order to manage and track our Agile process, JIRA is used to maintain the backlog of user stories and to track the current iteration. It will also be used to kept track of any issues found, giving it the ability to relate a bug to specific features and, thereby, a specific release.

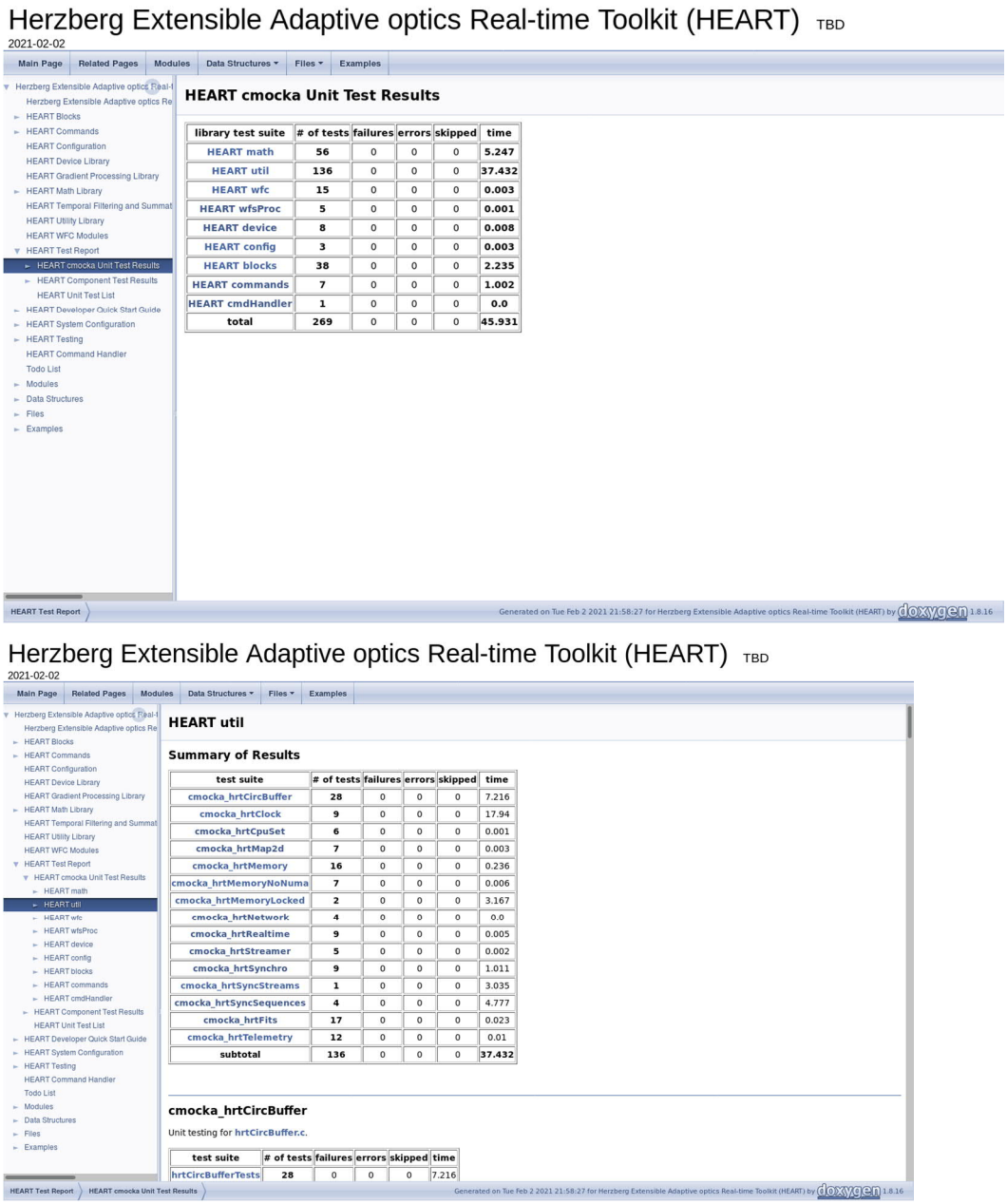
A value practice to Agile promotes the importance of keeping the client (i.e. Gemini) involved and aware of what is going on in the project. In fact, this greatly contributes to the power and success of Agile. Thus, using the Agile approach really lends itself to:

- Monthly iterations (some exceptions, e.g. December sprint combined with January due to the December shutdown at HAA). This is when the next releasable portion of work will be implemented.
- Monthly close-out and iteration meetings where discussions are held on:
    - what was done in the last iteration
    - additional stories added/changed to the backlog if new functionality is found or changes are necessary
    - what releasable features will be done during the next iteration
- Periodic code reviews
- Distributions of the JIRA sprint sheet for each iteration to ensure development is tracked.

6.1.3 Testing and Coverage

As mentioned previously, unit tests are written as the code is written. As part of the continuous software development practices, we utilize our Jenkins automation server such that those tests are automatically run on a continuous basis when a commit is made to the repository. It provides immediate feedback as to the success and quality of the build, in a well formatted output as shown in Figure 6-2.

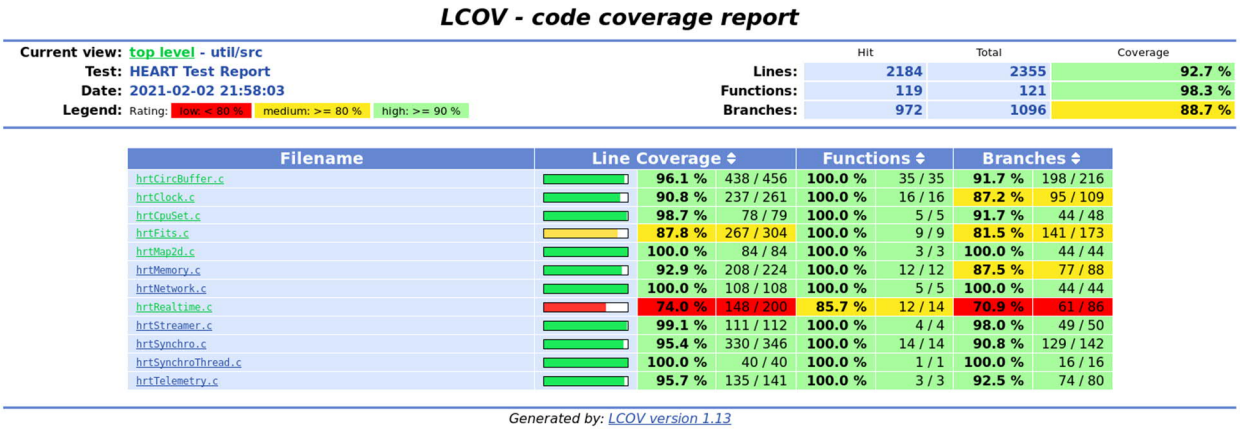
Figure 6-2 Unit Test output on existing HEART code



As part of this project, lcov is also run ensure that the code coverage of the units tests is satisfactory (Figure 6-3). Coverage reports indicate how much of the code is covered, the percentage of functions that are included, and that what logical code branches have been tested. While appearing extensive, it should be understood that there are limitations to the amount of

coverage that can be achieved. For example, many of the lines of code that are not executed are for handling error conditions that typically do not occur. Some only occur if there are much larger critical problems within the system. Consider a simple example where a program writes a small text file. The fopen() could fail (e.g. no write permission, not enough disk space, out of memory, etc.), the fprintf() could fail and the fclose() could fail. Each of those calls will have a return value which should be checked but, in order to recreate these errors, dramatic system conditions sometimes need to be created. This means that the "error case" is not executed by the normal code execution. This, in turn, means that the code for handling (or just noticing) the error is not executed and the coverage rate falls. This is where mocking can be useful, but there are limitations to the value of the extra effort to mock calls where it does not benefit the overall testing and quality of the code. Many of these cases are easily checked through standard code review practices or by simple code inspection.

Figure 6-3 Code Coverage Report



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - util/src	**Lines:**	2184	2355	92.7 %
Test:	HEART Test Report	**Functions:**	119	121	98.3 %
Date:	2021-02-02 21:58:03	**Branches:**	972	1096	88.7 %
Legend:	Rating: low: < 80 % medium: >= 80 % high: >= 90 %				

Filename	Line Coverage		Functions		Branches	
hrtCircBuffer.c	96.1 %	438 / 456	100.0 %	35 / 35	91.7 %	198 / 216
hrtClock.c	90.8 %	237 / 261	100.0 %	16 / 16	87.2 %	95 / 109
hrtCpuSet.c	98.7 %	78 / 79	100.0 %	5 / 5	91.7 %	44 / 48
hrtFits.c	87.8 %	267 / 304	100.0 %	9 / 9	81.5 %	141 / 173
hrtMap2d.c	100.0 %	84 / 84	100.0 %	3 / 3	100.0 %	44 / 44
hrtMemory.c	92.9 %	208 / 224	100.0 %	12 / 12	87.5 %	77 / 88
hrtNetwork.c	100.0 %	108 / 108	100.0 %	5 / 5	100.0 %	44 / 44
hrtRealtime.c	74.0 %	148 / 200	85.7 %	12 / 14	70.9 %	61 / 86
hrtStreamer.c	99.1 %	111 / 112	100.0 %	4 / 4	98.0 %	49 / 50
hrtSynchro.c	95.4 %	330 / 346	100.0 %	14 / 14	90.8 %	129 / 142
hrtSynchroThread.c	100.0 %	40 / 40	100.0 %	1 / 1	100.0 %	16 / 16
hrtTelemetry.c	95.7 %	135 / 141	100.0 %	3 / 3	92.5 %	74 / 80

Generated by: LCOV version 1.13

6.1.4 User Stories

The design of HEART and the custom portions of the Template and GNAO RTC include an exhaustive list of user stories/EPICs. Those were then combined into larger EPICs that are still exhaustive but are more functional specific and as a result will not change over time. These are the EPICs that effort is reported on and are listed in Table 6-1 along with the number of underlying epics/user stories. JIRA with the Agile extension is being used to contain the users stories, manage the sprints, and maintain the backlog. Any stories that are not completed, or need follow-up work gets moved to the backlog and is picked up in future iterations.

Table 6-1 - Major EPICs relevant to the Build

Section	# of Epics/User Stories
Framework	
Framework: Hardware Procurement	1
Framework: Initial hardware setup	1
Framework: Hardware setup custom	1
Framework: Hardware setup standard	2
Framework: Operating System Setup	3
Framework: Configuration Setup	1

Framework: Jira setup and maintenance	1
Soft Real-time	
SRT: Initial SRT interface	2
SRT: SRT Command interface	13
SRT: SRT Logging & Time Service	2
SRT: SRT Publish & Alarms	10
High Order Processing (HOP)	
HOP: Initial LGS Pixel Processing	3
HOP: LGS Test Server Pixel Processing	3
HOP: LGS Pixel Processing	3
HOP: Gradient Processing	3
HOP: HO MVM	2
HOP: SFS processing	3
HOP: TTF processing	1
HOP: Initial circular buffer	1
HOP: Final circular buffer	1
HOP: Initial RTS Streamer	1
HOP: Initial Synchronization Handler	2
HOP: HOP Thread Support: initial buffer, logging, synchronizing	5
HOP: HOP Thread Support: RPG Handler	2
HOP: HOP Support Threads: Performance Monitor, restructuring	2
Wavefront Correction Control	
WCC: Initial Synchronization Handler	2
WCC: Synchronization Handler	2
WCC: WCC Support Threads	5
WCC: Initial PWFS Processing	3
WCC: SRT interface	7
WCC: NFS Processing	8
WCC: SFS processing	1
WCC: LO Processing	2
WCC: Initial HO Post MVM Processing	2
WCC: HO Post MVM Processing	6
WCC: Initial DM Control	1
WCC: DM Control	4
WCC: TTS Control	2
WCC: Telescope Offloading	1
WCC: LGS TT Control	1
WCC: LGS TTF & Optimization	1
WCC: WCC support	1

WCC: NFS Testing	6
Telemetry	
TELM: TED Data Process	1
TELM: Initial RTS Support Processing	1
TELM: RTS Support processing	3
TELM: Disk Writer	1
TELM: RTC-RPG Interface	1
Test Control	
TEST: RTC Test Command Configuration	2
TEST: Storage of Test data	1
TEST: Send pixels	
TEST: Accept Commands	3
TEST: Send data	1
Assembly interface	
RTC-ASM: SRT Interface	5
RTC-ASM: Event publisher	5
RTC-ASM: RTC Assm: sim, mode, stop cmds	3
RTC-ASM: RTC Assm: pipeline cmd	4
RTC-ASM: RTC Assm: loop cmds	6
RTC-ASM: RTC Assm: RPG cmds	8
Reconstruction Parameter Generator	
SRT-RPG: Initial RPG processing	3
SRT-RPG: RPG Commands	8
SRT-RPG: RPG processing	10
Templates	
Templates: GN interfaces	4
Templates: GN template	3
Templates: GNAO template	4

6.2 MILESTONES AND FUTURE PHASES

The complete phase plan for the Common Code Base Realization Phase (as listed in the SOW) is the plan for the build of the Template RTC, GNAO RTC Implementation, and HEART.

The build timeline is very compressed and is detailed in [RD12] with the following functionality being developed along with their milestone dates.

Table 6-2 - Milestones associated with build

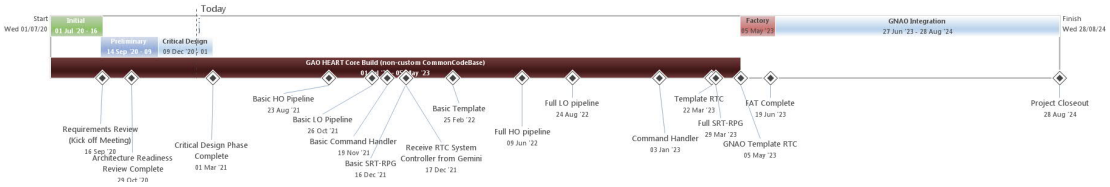
Task Name	Start
Basic HO Pipeline	Fri 06/08/21
Basic LO Pipeline	Mon 18/10/21
Basic SRT-RPG	Wed 08/12/21
Basic Command Handler	Wed 10/11/21

Basic Template	Thu 17/02/22
Full HO pipeline	Wed 01/06/22
Full LO pipeline	Tue 09/08/22
Full SRT-RPG	Tue 21/03/23
Command Handler	Fri 09/12/22
Template RTC	Tue 07/03/23
GNAO Template RTC	Thu 27/04/23
Receive RTC System Controller from Gemini	Thu 09/12/21

Internally there are monthly sprints, and Gemini requires that software release milestones shall be no less frequent than approximately once every four months. The build, which has already started for HEART, has already made a release to Gemini, and will continue to do so after the CDR. Over the scheduled remaining 26 months, there are 12 milestones currently scheduled.

This fits into the entire project as shown in the project plan snapshot shown in Figure 6-4.

Figure 6-4 Project Plan Snapshot



When the FAT plan is executed, this will not be performed until all tests are created and any updates to the FAT plan are complete. The tests will be written as the code is built and includes the component and integration tests. The planned activities for the FAT include:

- Preparing for the FAT
- Updating the OAI and OAT plans
- Executing the FAT at NRC
- Providing the report to Gemini.

It is recognized that the GNAO instrument will not be complete by the time the GNAO integration is schedule to happen. Until the contract is updated to reflect the new dates for the GNAO instrument, the schedule will remain as planned. Once GNAO is available and the date has been established, the Onsite Assembly and Installation (OAI) plan will be executed. This involves the delivery and installation of the hardware on site. After installation, a large part of the effort will be assisting the instrument specialist with integration. If an opportunity presents itself during the build of the RTC, an early version of the software will be sent before the OAI is executed. But this is dependent on the state of the GNAO instrument build. HAA will also assist Gemini with AO tuning with a final stage of executing the Onsite Acceptance Test (OAT) plan and completing the plan results report.

7. RISKS, BUDGETS AND TECHNICAL PERFORMANCE METRICS

7.1 RISKS

A risk register is maintained for this project and can be found in [RD6]. A summary of those risks are show in Table 7-1. The short timeline risk may be downgraded in severity once the overall

GNAO schedule has been re-baselined. However, this could lead to an increase in the risk severity related to the GNAO RFP being delayed, as this could results in more of the RTC software interface being developed before the rest of the system has been finalized. These risks will need to be re-evaluated after the GNAO design and schedule have been re-baselined, and the corresponding update to the GNAO requirements have been made.

Table 7-1 - Risk Register

Risk Description	Add. Details/ Consequences	Mitigation Plans	Overall Rating
GNAO: There is no RFP yet	[1] Need to make estimates not knowing about this [2] Need to negotiate with a contractor, it may be difficult, or they may choose equipment that will be difficult to incorporate		2
Interface to GNAO undefined	GNAO RFP has been cancelled	Find out if the hardware interface is non-changing and assume that interface	3
Timeline is very short and dependent on hardware systems that are unknown at this time	Computer hardware is selected as late as possible		9
Not yet in full agreement with the requirements	Deviation of the implementation	Work on a process to get the requirement changes accepted	9
Potential of other projects pulling resources away	Result in longer timeline	Review priorities as projects are added	9
SOW reflects thoughts for an unknown builder or plan and it provides no space for modifications to avoid unnecessary work	Work that does not add value to the project	Arrange contract changes to reflect the Agile build phase	6

NOTE: The descoping of the GNAO bench is not seen as a risk to the GNAO RTC, as the full MCAO functionality is required to satisfy the requirements for the Template RTC. The modification required to construct the GNAO RTC for the descoped Template RTC system are equivalent in nature to those required for the original system. Note included in this table are the 9 risks that have been retired.

7.2 DESIGN TRADES

There are no new design trades required to reduce risks. Previous HRT prototyping (described in Section 14) was done several years ago on hardware that was available at the time. It was shown to be adequate. Thus, the faster, newer hardware available should continue to be more than capable. Additional prototyping may be performed on the SRT system, however this effort would be to appropriately specify the minimum required hardware specification, since existing top-end hardware is more than sufficient.

7.3 FUTURE PHASE PLANS

The build phase has already begun for HEART and, as a result of the Agile development plan, slices of the system will be built. This means, that after CDR, the custom parts required for Gemini will also be worked on. Every iteration will be delivered with a test suit and Gemini is expected to build the code and run the tests. In the SOW, a Common Code Base Code (HEART) Review is included, but this will also include the Template RTC and GNAO RTC.

After a successful review, the FAT tests will be performed, having been previously written during the build and used for component and integration testing. These will also be delivered during the build. The output of the FAT test will trigger the creation of the FAT report and meeting to discuss the results will take place.

When the GNAO instrument is available, the RTC hardware will be delivered to GNAO and, after integration, the on-site acceptance test will be performed.

All of these phases are detailed in the project plan [RD12].

8. TECHNICAL PERFORMANCE METRICS

Technical Performance Metrics (TPMs) show how well the RTC satisfies its high priority requirements (typically less than 1% of requirements should have TPMs). It also provides assessment of those requirements when designing, implementing and testing.

The TPM requirements can be split into two categories, the first being the key system performance budget requirements shown in Table 8-2. These requirements have been initially verified by prototyping activities, as discussed in Section 14, and will be ultimately verified using the Template RTC during acceptance testing. The second category are those requirements that fundamentally drive the design of system, shown in Table 8-1. These requirements are primarily verified by design at the various design phases of the project and will again be ultimately verified by demonstration with the Template RTC.

Table 8-1 - Technical performance Metrics by design

#	Requirement
8.2.1.1	The RTC System shall accept pixels from a maximum of three NGS WFS, with a goal of 6 NGS WFS via an Ethernet-based interface.
8.2.1.15	The RTC System shall calculate wavefront tip-tilt corrections based on the selected TT NGS WFS centroids.
8.2.1.16	The RTC System shall send wavefront tip-tilt corrections to the Tip-Tilt Mirror Controller.
8.2.2.1	The RTC System shall accept pixels from a minimum of one and a maximum of 5 LGS WFS simultaneously via an Ethernet-based interface.
8.2.2.7	The RTC System shall calculate the LGS centering error for each selected LGS WFS individually.
8.2.2.8	The RTC System shall send the LGS centering errors to the corresponding Laser Fast Steering Mirror Controller via an Ethernet-based interface.
8.2.2.13	The RTC System shall calculate the deformable mirror shapes required to correct turbulence at each altitude, including plate scale modes.

8.2.2.16	The RTC System shall send the calculated Deformable Mirrors shapes to the corresponding Deformable Mirror via an Ethernet-based interface.
8.2.3.1	The RTC System shall optimize, at a minimum, the performance of the following processes: NGS Gradient Calculation; NGS Reconstructor; Low Order Temporal Filter; Tip Tilt Servo; LGS Gradient Calculation; LGS Reconstructor; High Order Temporal Filter; DM Servo
8.2.4.1	The RTC System shall generate real-time telemetry data.
8.2.4.6	The RTC System shall store telemetry information internally. Pixel data shall be store for one full night, slopes and DM commands shall be stored for 30 days, and intermediate computed values shall be stored for 8 days. This assumes 6 hours per night of operation and default decimation rates.

All of the requirements are also tracked as part of the requirement compliance, updated at each phase of the design. Both of these categories of requirements are ultimately verified by the Template RTC, making the successful demonstration of the Template RTC in operation the definitive TPM.

Benchmarking has been done as part of the NFIRAOS development and the amount of time allocated for reading, calibrating, change to gradients and HO MVM of the pixels was 500 usec (see the SPIE paper which shows the breakdown of the estimated latency [RD9]), This, of course, is dependent upon the size of the WFS(s) and the speed that the WFS controller can deliver those pixels. The RTC maximum latency is measured from the arrival of the last WFS pixels to the output of WFC. But, because the GNAO HO WFS the pixels are processes as they are received, as long as the RTC can keep up with the pixel arrival rate (reading, calibrating, gradients, HO MVM), the readout time is not as important.

Prototyping was done using one core on a development virtual machine that can do the HO MVM (416 rows, 768 columns) in 60 usec and the "Template" RTC (1116 rows, 768 columns) MVM (which increases from 1DM to 3 DMs) takes 150 us. The times above are for a single LGS WFS as all LGS WFS are run in parallel (i.e. there is no need to sum the result vectors at the end). If each high order MVM uses N cores, then the speed increase will be quite close to N for a small number of cores. If 4 cores are used per MVM, then there are only using 20 of the possible 64 CPU cores for five LGS WFS. The expectation is that the specified AMD EPYC will be significantly faster than the development virtual machine that was used for the test.

Figure 2 in the RTC prototyping paper [RD9] shows a breakdown of the estimated latency. The estimates in the prototyping paper have the last actuator being sent by about 920 usec after the first pixel arrival. For GNAO, this would be reduced by ~60 usec due to less time being spent adjusting for clipping, with another ~80 usec reduction (as the DM vectors are not sent between servers). It will also result in another ~40 usec reduction to send DM vectors to DME and ~50 usec less for finishing up the MVM. Therefore, the last actuator should have been sent within ~190 usec after the last pixel, with an addition 10 usec used for contingency.

The requirements state a maximum latency of 500 usec for the RTC to send out corrections. The maximum frequency for each path is 1kHz. The budget is detailed below.

8.1 TIMING BUDGET

It is important that the HRT loop can run within the key requirements. Table 8-2 details the system budgets (rates, latency, jitter) for the Template RTC and GNAO Template RTC. This budget is critical for the system performance. Section 14 contains details to show that the system budgets can be met.

Table 8-2 - Key System Budget Requirements

#	Requirement
8.3.1.1	NGS rate < 1kHz SFS rate < 10 Hz
8.3.1.2	NGS max latency 1ms, with a goal of 0.5ms.
8.3.1.3	NGS max jitter over 60s, of 100 us std dev (from last pixel)
8.3.2.1	LGS rate < 1kHz, goal <2kHz.
8.3.2.2	LGS max latency 500us
8.3.2.3	LGS max jitter over 60s, of 100us std dev (from last pixel)

The RTC maximum latency is measured from the arrival of the last WFS pixels to output of the Wavefront Correction. To do this, we first look at the Template RTC block diagram (Figure 8-1) and LGS pixels coming in first. As the pixels are read (Figure 8-1, light blue box), they are processed and the MVM is executed (Figure 8-1, red box). Therefore, after all pixels are read in, there is only 1 more group of pixels to process and perform the MVM on (Figure 8-2, point A). After all of the LGS's HO output vectors are combined, they are then synchronized (Figure 8-1, brown box). This is the section where there is the maximum amount of delay (Figure 8-2, point B) in waiting for all of the pixels to arrive for all LGSs. After that point, the frame from that particular WFS will be dropped. Finally, the Closed Loop wavefront correcting (Figure 8-1, yellow box) occurs where the vectors are conditioned and sent to the output devices. There are fewer NGS pixels to be sent and, therefore, all pixels are received (Figure 8-1, green box, and Figure 8-2, point D) before they are processed and the MVM performed. Similar to the LGS pixels, the NGS's LO output vectors are also synchronized (Figure 8-1, brown box).

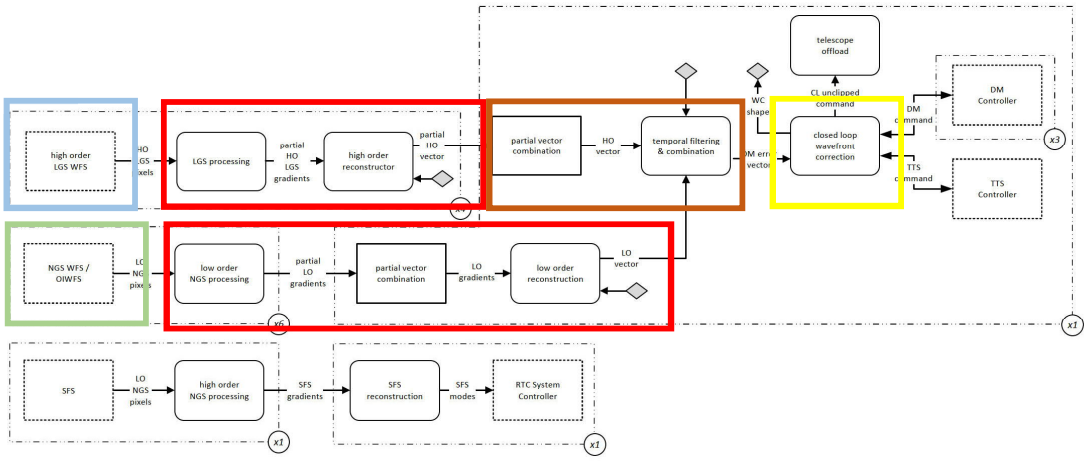


Figure 8-1 Template Block Diagram showing timing blocks

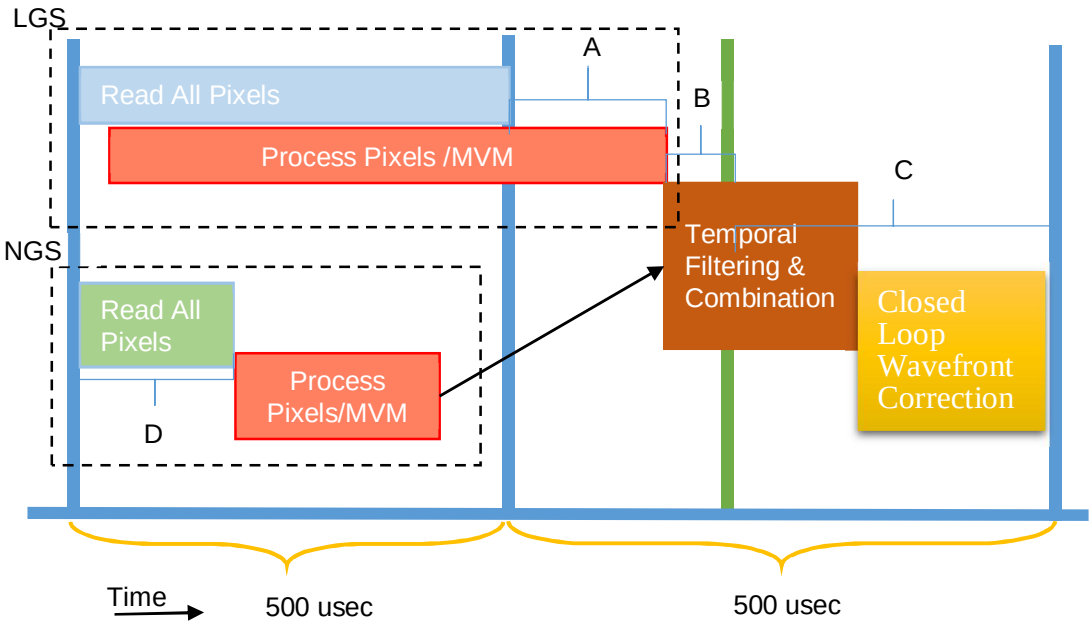


Figure 8-2 - Timing Diagram

A benchmark to measure timings for the LGS path is described in Section 14, which includes: (i) transmission of 5 x LGW WFS pixels from a test computer over 500 µs at a frame rate of 1000 Hz over two 10 GbE ports to a second RTC test computer; (ii) reception, processing of pixels (calibration and gradients), partial MVMs for the 5 LGS, and combination into a DM vector; and (iii) transmission of the DM vector for each frame back to the first test server so as to measure round-trip times. This encompasses A, and portions of B and C above. These results are further augmented in Table 8-3 to estimate the small contributions of other tasks not included in the round-trip benchmark. To summarize, we estimate that wavefront corrector commands can be delivered within ~150 µs following the reception of the last LGS WFS pixel. We are therefore very confident in the budget presented in Table 8-3 which assumes 300 µs to process pixels into

gradients, with 100 µs for additional tasks and delivery of commands to wavefront correctors, and 100 µs of contingency.

Table 8-3 – Timing Budget

Budget	LGS	NGS
Max time to read the WFS pixels	250usec> and <500 usec (blue box)	500 usec (green box)
Max time to process pixels (change to gradients), perform MVM)	300 usec (red box) *After receiving last pixel*	150 usec (red box) *After receiving all pixels*
SubTotal (Max time to wait for all HO/LO Vectors (sync point) *After this point both paths are synced*	**800** **(B)**	
Close loop Wavefront correcting (output to DM/TT)	100 usec (C)	
Contingency	100 usec	
Total	**1000 usec**	**1000 usec**
Latency *(max time after last pixel to complete processing)*	500usec	1000usec
Deviation over 60 seconds	100usec	100usec

9. TEMPLATE RTC SOFTWARE COMPONENTS

The HEART design is used to create the Template RTC, which has a SRT System, HRT System, Telemetry Handler and Command Handler. The HRT block diagram is shown in Section 2.1.1, Figure 2-2 and repeated here in Figure 9-1. The HRT blocks are put into pipes which are configured to trigger when the pixels are sent to the RTC. The inputs and outputs to each block are shown in block diagrams in the following sections. The format of the inputs and outputs are all circular buffers which are detailed in the Internal ICD [RD14] Circular Buffer section, with the Pipeline details in the Pipeline section. The processing steps are detailed in the HEART design document [RD1], with any deviations from that design called out below. The interfaces between any other components are also explained below, with the formal format for those interfaces detailed in the External ICD [RD2]. As with the HRT, the SRT, Telemetry Handler, and Command Handler exceptions are detailed in Sections 5.4, 5.5 and 5.2.

This Template RTC is an example of how to use HEART to create a MCAO RTC. There are example templates within the HEART code base to show how to create different types of AO systems.

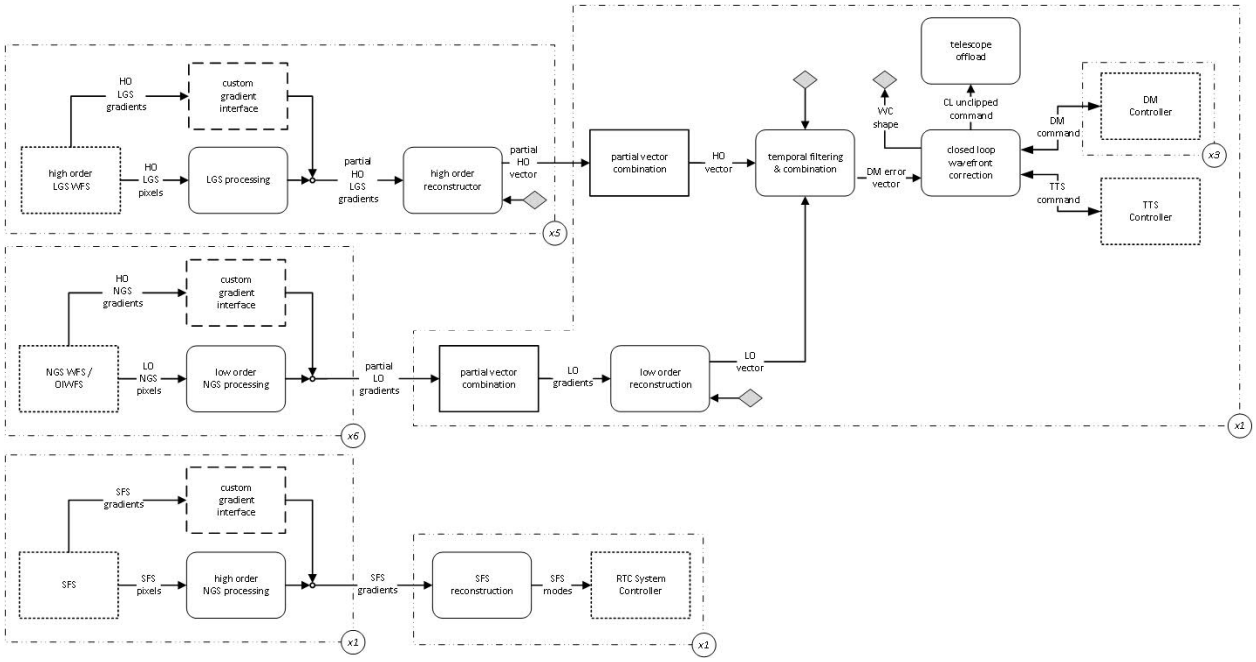


Figure 9-1 - Template RTC HRT block diagram built with HEART

The Template RTC HRT pipeline, as shown in Figure 9-1, is nominally modelled as a prototypical MCAO system. Five LGS WFS pixel streams are processed (labelled “LGS processing” in Figure 9-1) and reconstructed in parallel (“high order reconstructor”). They are then summed (“partial vector combination”) to produce a total High Order error vector (HO) (“partial vector combine”). Up to six NGS WFS pixels streams, including those from On-Instrument Wavefront Sensors (OIWFSs), are also processed in parallel (“low order NGS processing”) and the aggregated gradients are used to reconstruct (“low order reconstruction”) a Low Order mode vector (LO vector – includes tip, tilt and plate scale modes). The total high order error vector is filtered and the low order mode error coefficients are filtered and projected into DM command space to be summed with the high order error vector (temporal filtering & combination). The net error vector is integrated and then decomposed into three virtual DMs, corresponding to different altitudes (“closed loop wavefront correction”). The ground-layer DM is further decomposed into DM commands and tip/tilt commands. These commands are filtered and resampled for telescope offloading (“telescope offload”). Slow focus sensor (SFS) pixels or gradients are processed (“high order NGS processing”) to compute focus, and other modes. These modal coefficients (SFS modes) are offloaded to the Systems Control (“SFS reconstruction”), which will be able to take action accordingly (i.e. refocus the LGS WFSs and/or modify LGS WFS reference slopes).

Of the HEART blocks shown and detailed in the HEART design document [RD1], the following HEART blocks are not used in the Template or GNAO RTC:

- Figure Processing
- Figure Reconstruction
- Figure Post-Processing
- High Order NGS Truth Processing
- High Order Truth Reconstruction
- Open Loop Wavefront Corrector Control
- POL gradients to the Telescope

For the remaining blocks, there are small parts of the HEART blocks that are not needed by Gemini. Those differences are highlighted in the Figures below with grey boxes and detailed in the following sections.

9.1 CONFIGURATION FILE

Details on the format of the configuration file, and how it is read, are covered in the HEART document [RD1]. Details on what blocks use the configuration information are found in the Internal ICD [RD14]. There is only one configuration file for a single RTC from which each block absorbs that necessary information.

Upon startup of the RTC, the configuration information is loaded from a file, and any files that are referenced are loaded. All files are kept under revision control along with the source code. For the Template RTC, the custom configuration parameters are listed in Table 9-1.

Table 9-1 - Template RTC Custom Configuration Parameters

Keyword	Description of the value
Custom for Gemini:	
SFS_WFS_SIZE	number of HO sub-apertures per WFS, and number of slopes
SFS_NUM_MODES	Number of Modes

The standard HEART configuration parameters are described in Section 3.1 of the HEART Internal Interface Document [RD14].

During build and integration there will also likely be experimentation with the core and node assignments for worker threads and CBs to optimize performance for the specific final configuration of GNAO.

The configuration that is specific to the Template RTC ICD is shown in Table 9-2.

Table 9-2 - Template RTC Specifications

Specification	Value
number of LGS WFSs	5
number of NGS WFSs	Up to 6
Number of Modes	10
Number of HO sub-apertures per WFS, and number of slopes	22x22, ~3800
Number of LO sub-apertures per WFS, and number of slopes	1x1
Number of LO Modes	6
Number of DMs	3
Number of actuators per DM	393,321,321

9.2 STANDARD INPUT BLOCKS

The standard input blocks are used to read either pixels or centroids. There is a standard block for each WFS. Each of them have custom handler interfaces, and for the Template RTC they are:

- 5 LGS WFS
- 6 NGS WFS
- 1 SFS WFS

The Standard WFS Input block calls a custom handler function to trigger the downstream block at the start of the arriving data for a frame. The custom handler function is defined in the External Interface document [RD2]. For the Template RTC, the custom handler will be receiving the data from an external client WFS tester (discussed in Section 5.6) which will send the pixels using a socket interface. The format of the datagram will be a 2D image, with the X/Y (0,0) location corresponding to the bottom left hand corner.

The custom handler reads in the input stream, stores it in memory in the format specified by the cbXXXRawPixels (where XXX is LGS, NGS or SFS) circular buffer. The Standard input block will save the data to the appropriate circular buffer and inform the next block when the data is available. The custom handler will be repeatedly called until the entire frame is read, repeating until the Input WFS block is instructed to stop. The block configuration, available to the custom handler during initialization, will include information such as the frame size for each frame.

The custom handler will read the pixels/gradients and put the amount of expected data into the destination pointer location. If the incoming pixel stream does not match the dimensions, then the reading function will read until the total amount is reached for the frame and store the data within the provided buffer. After each data chunk is read and returned, the progress data structure is updated.

The format of the data received from the client input tester is detailed in Table 9-3.

Table 9-3 – Input WFS Pixels Datagram Format (UDP)

Size (bytes)	Data Type	Description
2	uint16_t	WFS source identifier
2	uint16_t	N_{pix} : Number of pixel (gradient) values included in current datagram.
2	uint16_t	Datagram sequence number within WFS frame. The sequence number is reset to zero for the first datagram of each WFS frame.
2	uint16_t	Number of datagrams per WFS frame.
2	uint16_t	Full image width in pixels (gradients).
2	uint16_t	Full image height in pixels (gradients).
2	uint16_t	Datagram image width in pixels (gradients).
2	uint16_t	Datagram image height in pixels (gradients).
4	uint32_t	Raster scan pixel (gradient) index for first pixel (gradient) within datagram.
4	uint32_t	WFS frame number The frame number will be incremented by 1 every time the WFS starts imaging in response to a frame trigger signal.
8	uint64_t	Timestamp (nanoseconds since epoch).
2 (or 4) * N_{pix}	uint16_t or float	16 bit unsigned pixels or 32 bit floats (e.g. centroids) within datagram are sent in raster scan order. Packed 12 bit pixels are supported which allows storing two pixels per three bytes. These will be unpacked into 16 bit unsigned integers.
4	uint32_t	32 bit checksum

9.3 STANDARD OUTPUT BLOCKS

Standard output devices for the Template RTC are:

- Deformable Mirror (DM) controllers,
- Tip/Tilt Stage (TTS) controllers,
- Laser Guide Star Fast Steering Mirror (LGS FSM) controllers,
- Telescope offloading
- and external computing systems which accept externally streamed telemetry (e.g. for user interfaces or offloading low order modes to the secondary control system).

There are standard blocks for each of these devices. Normally each of them have custom handler interfaces, and is defined in the External Interface ICD. For the Template RTC the interfaces and protocol will only be the interface listed in the External Interface ICD.

The Standard DM and TTS Output block calls a custom handler function that is triggered by the upstream block when the commands are ready. The custom handler function is defined in the External Interface document [RD2]. For the Template RTC, the custom handler will send the data to an external client DM and TTS tester (discussed in Section 5.6) which will read command using a socket interface.

The custom handler reads the DM and TTS commands stored in memory in the format specified by the cbDm and cmTt circular buffers. The Standard output block will read the commands from memory and write them out the socket interface. The block configuration, available to the custom handler during initialization, will include information such as the DM vector size for each frame.

The format of the data received by the DM and TTS client output testers are detailed in Table 9-3 and Table 9-4 respectively.

Table 9-4 – Output DM Datagram Format (UDP)

Size (bytes)	Data Type	Description
2	uint16_t	DM target identifier.
1	uint8_t	Datagram sequence number within DM vector. The sequence number is reset to zero for the first datagram of each DM vector.
1	uint8_t	Number of datagrams per DM vector. The limit of 255 datagrams per DM vector is not an issue since over 300 actuator values can be sent within a single non-jumbo datagram.
2	uint16_t	Actuator index offset. DM actuator index of first actuator included in datagram. The use of 16 bit actuator indexing allows over 65000 actuators per DM.
2	uint16_t	Number of actuator values included in current datagram.
4	32 bit integer	RTC frame number.
4 * N	32 bit floats	One floating point value for each actuator value sent. Actuator order is dependent upon the DM controller. Units are in microns
4	uint32_t	CRC-32C checksum

Table - Output TTS Datgram Format (UDP)

Size (bytes)	Data Type	Description
4	*uint32_t*	Frame number
4	float	Tip command (milli-radians rotation about axis #1)
4	float	Tilt command (milli-radians rotation about axis #2)

4	uint32_t	Simple 32 bit checksum

All other output interfaces and datagram formats are described in section 3.2 of the HEART External Interface Document [RD2].

9.4 LGS PROCESSING BLOCK

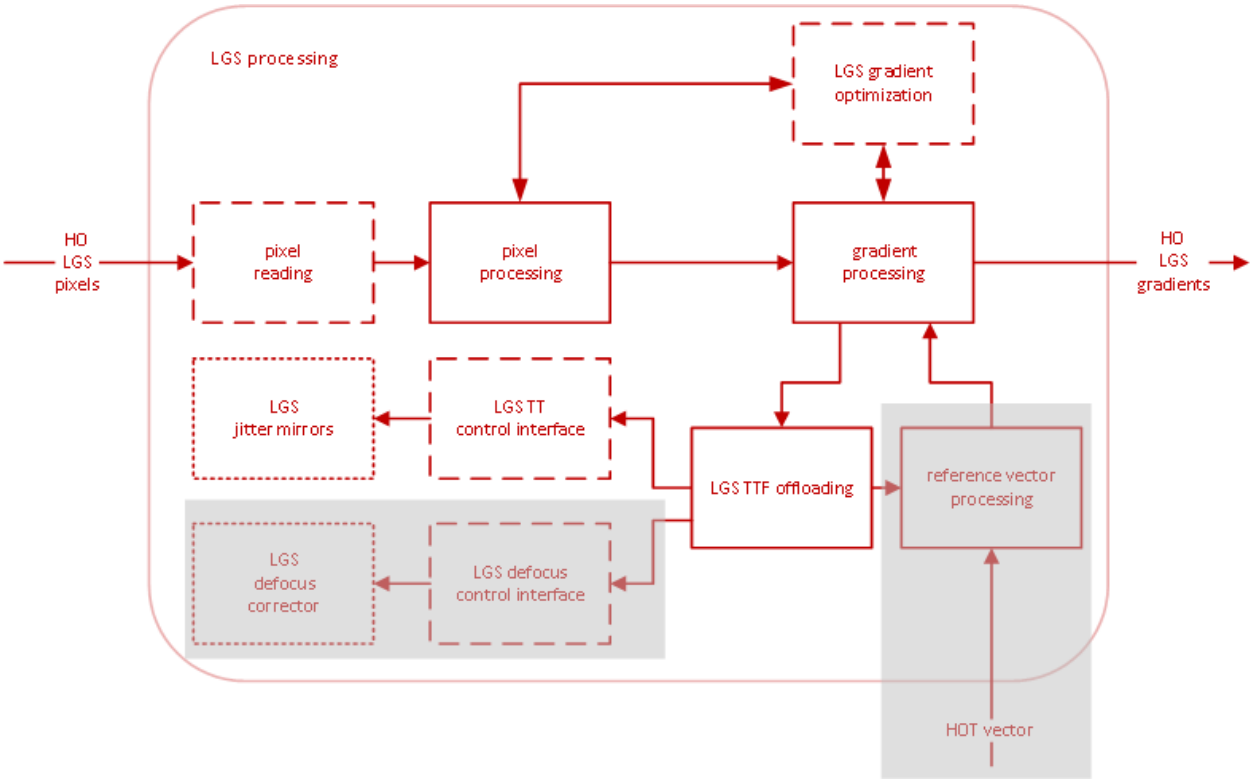


Figure 9-2 – LGS Processing Block Diagram

For the LGS Processing Block, the inputs are the HO LGS pixels. For the Template and GNAO Template RTC, there will be multiple blocks, one for each LGS. If the detector allows it, as the chunks of data are received, the data will be processed into gradients. For further details on the processing and data, refer to the HEART Design document [RD1].

The Gradient Processing sub-block also accepts LGS reference vector updates from the Reference Vector Processing sub-block, although this data flow is not required for Gemini and has been greyed out in the above figure.

Similarly, the focus modes from the gradient vector are sent to the custom LGS Defocus Control Interface sub-block, which performs any control processing required, and transmits the mode to the appropriate sub-system, which are also greyed out. This focus correction path is not used by Gemini, as LGS WFS focus correction is handled by the System Controller via the custom SFS path.

9.5 HIGH ORDER RECONSTRUCTION BLOCK

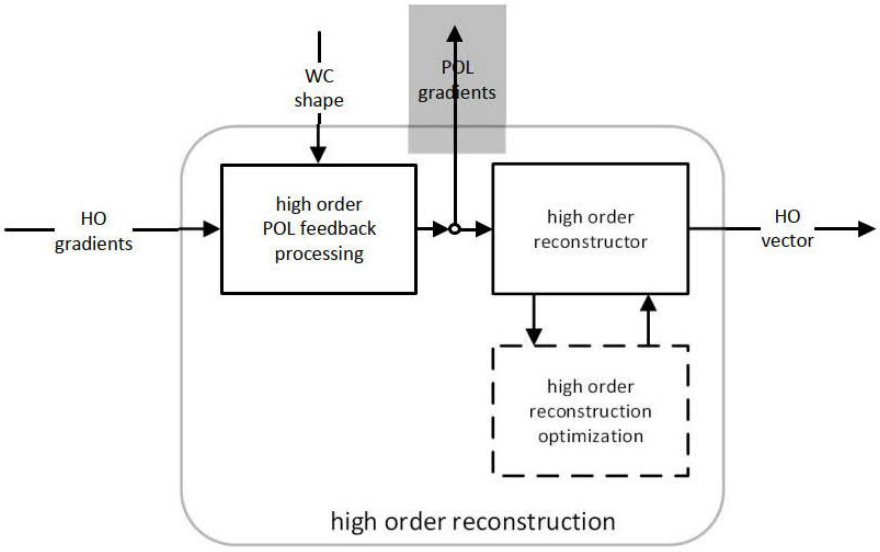


Figure 9-3 – High Order Reconstruction Block Diagram

In the High Order Reconstruction Block gradients from the LGS Processing Block stream into this block. They are added to previous WC shape projected into gradient-shape to create the POL gradients. The POL gradients are then multiplied by the reconstructor matrix. Once all gradients have been multiplied by the matrix, the resulting partial HO vector is sent to the next block. This is a partial HO vector since it is only the part of the overall HO vector, corresponding to the given WFS. This is covered in the HEART design [RD1].

Additionally, once all the POL gradients have been computed, they can be sent to the Telescope Offload block to correct for the primary mirror scalloping mode. However, this connection is not needed by Gemini and has been greyed out.

9.6 LOW ORDER NGS PROCESSING

This block does not diverge from the HEART design of the block. The block diagram is included as a reference to show inputs and outputs, but the processing is covered in the HEART design [RD1].

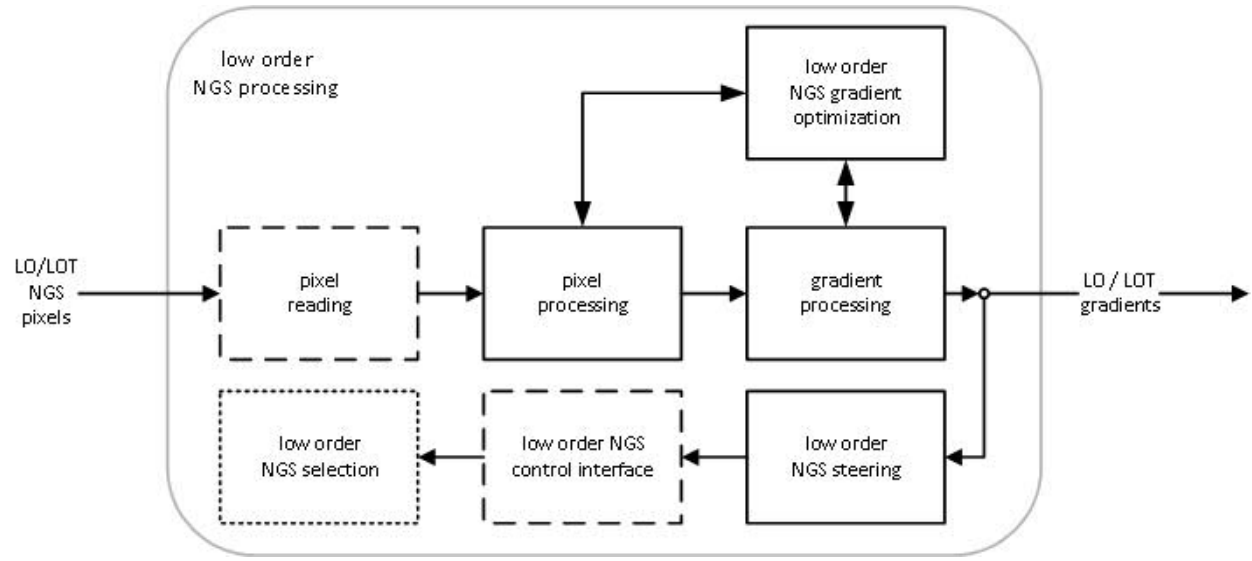


Figure 9-4 – Low Order NGS Processing Block Diagram

For the Template and GNAO Template RTC, the custom low order NGS control interface will be configured to send pupil centering error to the system controller.

9.7 LOW ORDER RECONSTRUCTION

This block does not diverge from the HEART design of the block. The block diagram is included to show inputs and outputs but the processing is covered in the HEART design [RD1].

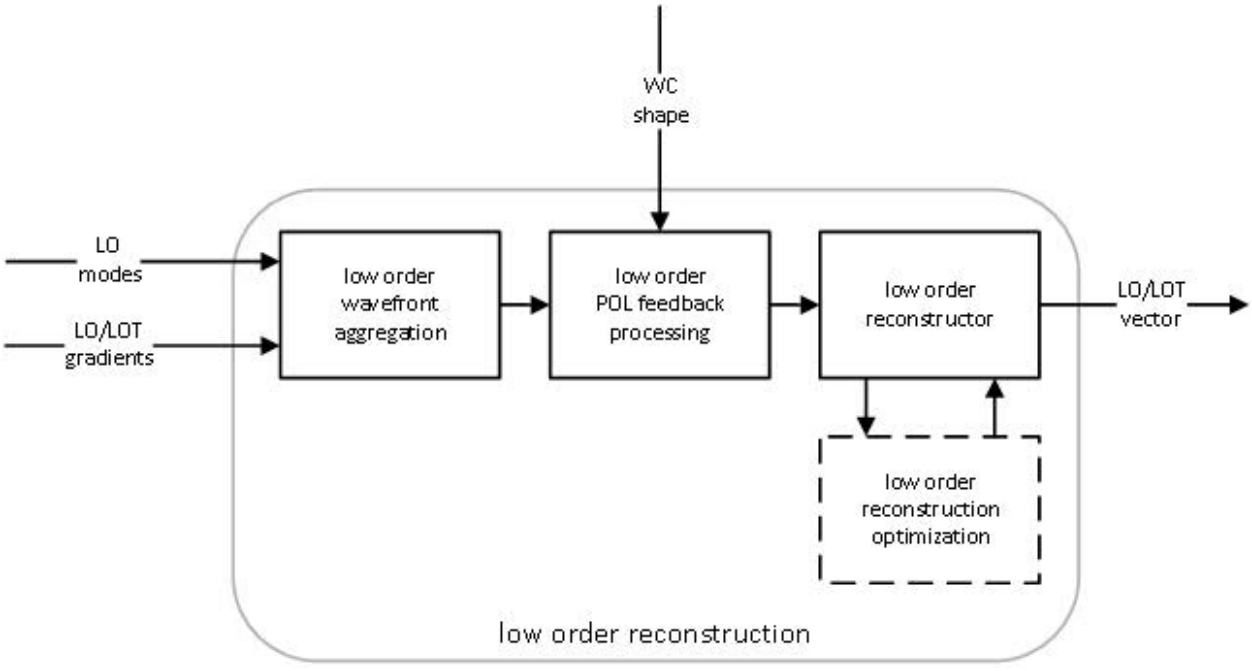


Figure 9-5 – Low Order Reconstruction Block Diagram

9.8 PARTIAL VECTOR COMBINATION

The Partial Vector Combination blocks are only required if there are multiple High Order or High Order Truth Reconstruction blocks. They simply aggregate partial HO/HOT vectors from the Reconstruction blocks to produce a complete HO/HOT vector. The design is detailed in the HEART design document [RD1].

In the case of Template and GNAO Template RTC, there is no HOT WFS, therefore this block only aggregates partial HO vectors from each of the LGS WFS HO Reconstructions blocks.

9.9 TEMPORAL FILTERING & COMBINATION

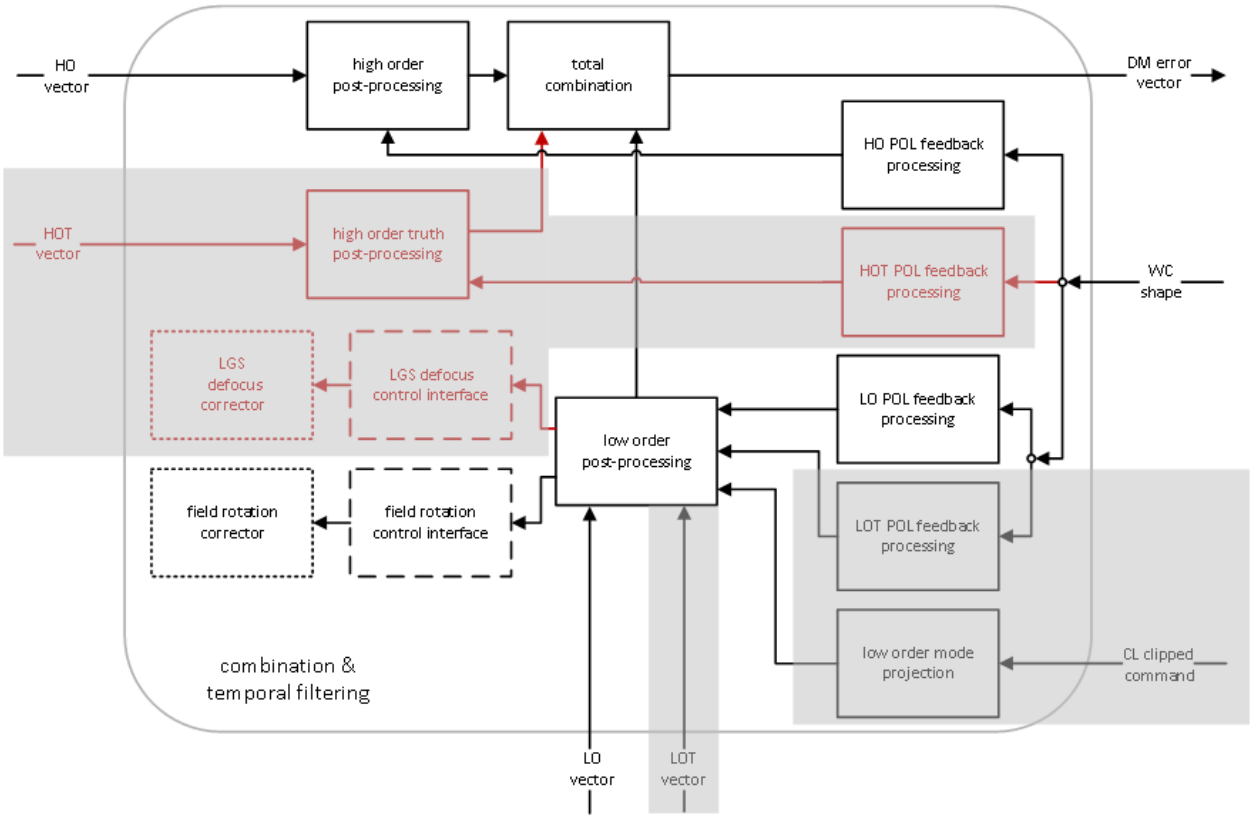


Figure 9-6 – Temporal Filtering & Combination Block Diagram

The Temporal Filtering & Combination block, as shown in Figure 9-7, accepts HO, HOT, LO and LOT vectors from the corresponding reconstruction blocks, as well as WC shape and closed loop clipped command from the Closed Loop Wavefront Correction block. The sub-blocks and connections that have been greyed out are not required for Gemini, namely those blocks associated with the HOT vector, LOT modes, and LGS defocus offloading.

The High Order Post-Processing and High Order Truth Post-Processing sub-blocks subtract the POL feedback vector from the HO/HOT vectors to produce and error vectors, completing the POL feedback processing. In addition, a temporal filter is applied to the vectors before they are passed to the next block. . The same temporal FIR filter is used for all the elements of each vector.

The HO and HOT POL Feedback Processing sub-blocks, process the WC shape vector so that it can be subtracted from the HO/HOT vectors.

The LO and LOT POL Feedback Processing sub-blocks operate similarly to the HO and HOT equivalents, except that the WC shape is also projected into the LO and LOT modes. Similarly, the Low Order Mode Projection sub-block can project the clipped closed loop commands into LO modes for advanced filtering, such as Kalman filtering, which is not required for GNAO.

The Low Order Post-Processing sub-block merges the LO and LOT paths into a single LO error vector. Prior to the merge, the POL feedback is subtracted from the LO and LOT vectors to produce error modes. These two error mode vectors are temporally blended. A high pass filter is applied the LO path. A Zero Order Hold (ZOH) and scaling factors followed by a low pass filter are included in the LOT mode to account for the rate difference between the LO and LOT paths. The LO and LOT modes are then summed together to create the net LO modes. In the case of Gemini, there is no LOT path, so this step simply allows the LO path to be passed through unaffected. Then each modal coefficient in the net LO error vector has a temporal FIR filter applied. The framework also supports more sophisticated temporal filtering methods, such as Kalman filters, which requires the clipped closed loop commands feedback be projected into LO modes, so that the filter state can be adjusted in case of clipping. However, this style of filter is also not required for Gemini.

The Low Order Post-Processing sub-block extracts the field rotation from the net LO mode vector and sends it to the System Controller. Additionally, if the field distortion (plate scale) modes are disabled, then they will be zeroed at this point, prior to being injected into the HO path.

Additionally, the Low Order Post-Processing sub-block can compute focus modes that can be offloaded via the custom LGS Defocus Control Interface sub-block. However, this is not required by Gemini, as this is handled by the SFS path.

Finally, the Total Combination sub-block adds the HO and HOT error vectors, however in the Gemini case the HOT is not used, therefore this is a no-op. The net LO modes are then mapped into the HO error vector-space, via the multiplication by a projection matrix, and then added to the HO/HOT error vector to produce the final “DM error vector”.

9.10 CLOSED LOOP WAVEFRONT CORRECTION

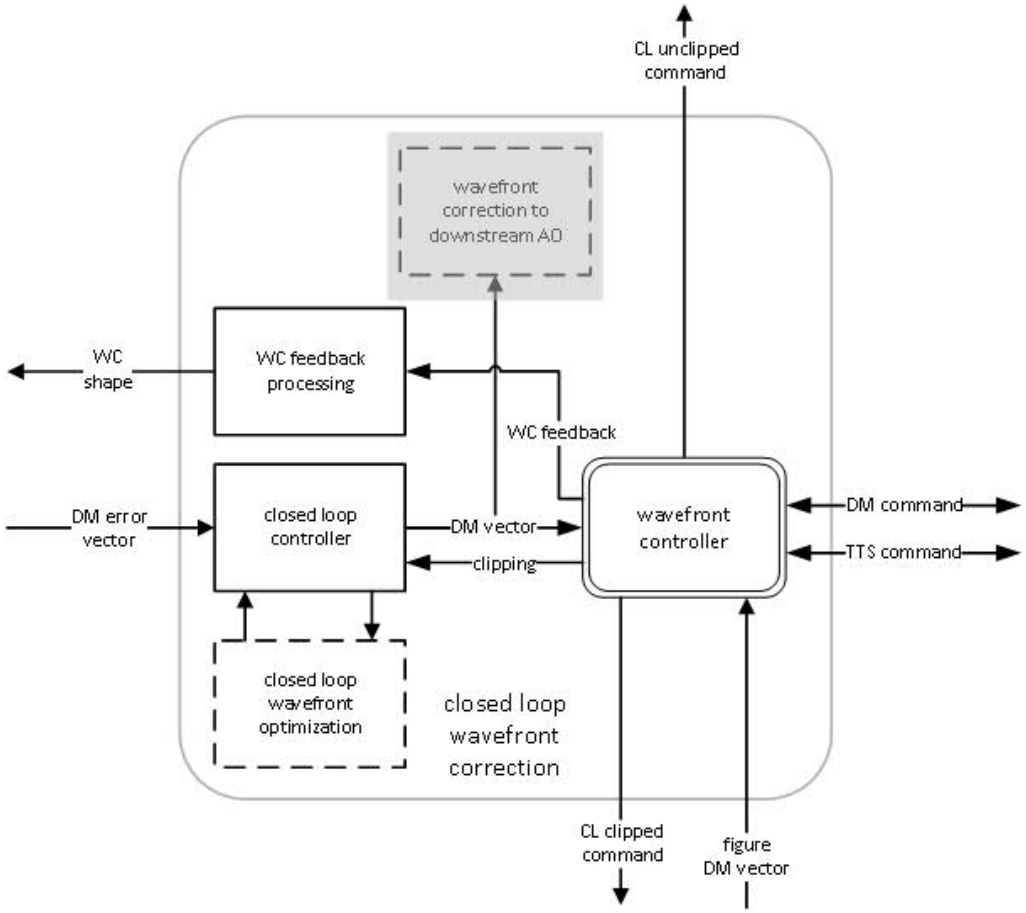


Figure 9-7 – Closed Loop Wavefront Correction Block Diagram

The Closed Loop Controller sub-block, applies an integrator to the DM error vector to produce the DM command vector. The DM command vector is then split into one or more virtual DM command vectors and sent to corresponding Wavefront Controller sub-blocks. The Closed Loop Controller sub-block removes uncontrollable modes from the integrator (cleanup path) and adjusts the integrator based on clipping feedback to avoid windup.

The Wavefront Controller sub-blocks then decompose the virtual DM command vector into DM command vector(s) for the physical DM and tip and tilt commands for the TTS. This block then triggers the DM and TTS Standard Output blocks discussed in Section 9.3.

The downstream AO connection is not strictly required by the Gemini system, but will be included in HEART to support future MOAO systems and will be incorporated into the Template as part of MOAO development.

9.11 TELESCOPE OFFLOAD

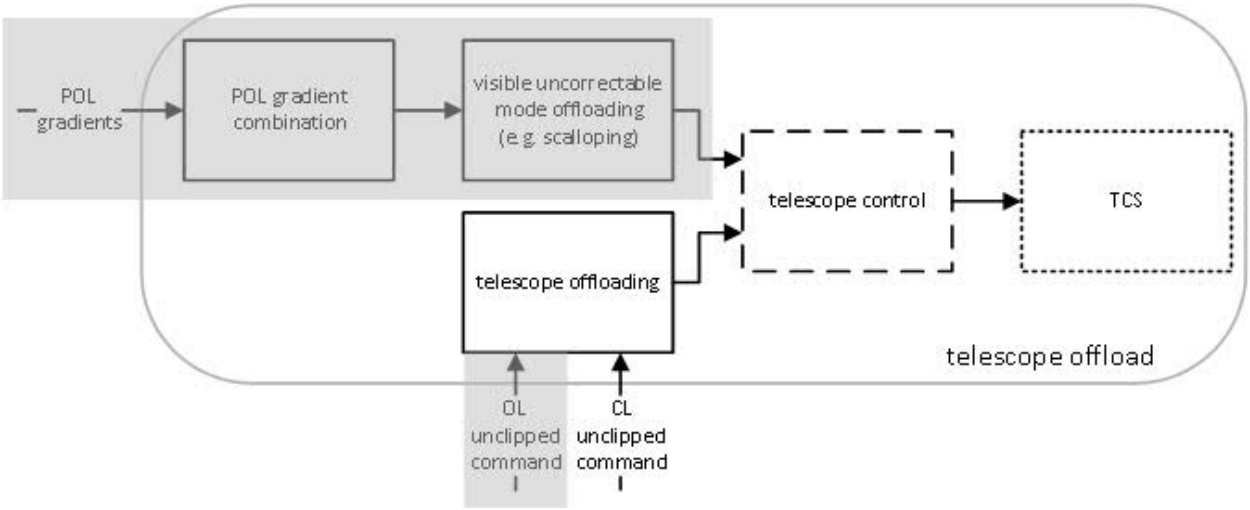


Figure 9-8 – Telescope Offload Block Diagram

The POL Gradient Combination sub-block aggregates POL gradients from all of the High Order Reconstruction blocks to produce a complete POL gradient vector. This vector is projected onto visible yet uncorrectable modes, such as primary mirror scalloping modes for a segmented telescope. These modes are sent to a custom Telescope Control sub-block which communicates with the appropriate sub-system within the observatory. This is not required by the Gemini system.

For Gemini, only the CL unclipped command path is required in the Telescope Offloading sub-block, which projects the commands to a set of modes which are low pass filtered and then down-sampled to a suitable rate to be offloaded. The down sampled modes are sent to the custom Telescope Control sub-block which communicates with the appropriate sub-system. For Gemini, this would be the offloaded coma and focus corrections and primary mirror modes to the System Controller and also includes tip and tilt sent directly to the Secondary Control System. The rest of the processing for this block is detailed in the HEART design document [RD1].

9.12 SFS RECONSTRUCTOR – CUSTOM FUNCTIONALITY

The Slow Focus Sensor (SFS) loop is independent of the High- and Low-order loops discussed above, and is specific to the Gemini system. The SFS loop will utilize the information supplied by the SFS to perform its calculations and determine the SFS modes (primarily the focus mode, but may include other higher order modes). Using the gradients computed from the SFS pixels, the RTC will calculate the SFS modes which will, in turn, report the result back to the System Controller to make the necessary adjustments to the LGS defocus mechanism and any other relevant sub-system.

10. GNAO TEMPLATE RTC SOFTWARE COMPONENTS

The HEART design is used to create the GNAO Template RTC, which has a SRT System, HRT System and Command Handler. The Template RTC HRT block diagram is shown in Figure 2-2 with the modified GNAO Template RTC block diagram listed below in Figure 10-1. The GNAO Template RTC is effectively the same as the Template RTC, except with different dimensions and parameters, mainly with one fewer LGS WFSs and two fewer DMs. The inputs to each block are shown in each block diagram in the following sections. The format of inputs and outputs are all circular buffers which are details in the Internal ICD [RD14]. The processing steps are detailed in

the HEART design document, and where there are deviations from that design they are called out below. The interfaces between any other components are also called out below, and the format for those interfaces are detailed in the External ICD [RD2]. As with the HRT, the SRT, Telemetry Handler, and Command Handler exceptions are detailed in Sections 5.4, 5.5 and 5.2.

This GNAO Template RTC is an example of how to use HEART to create a LTAO/GLAO RTC. There are example templates within the HEART code base to show how to create different types of AO systems.

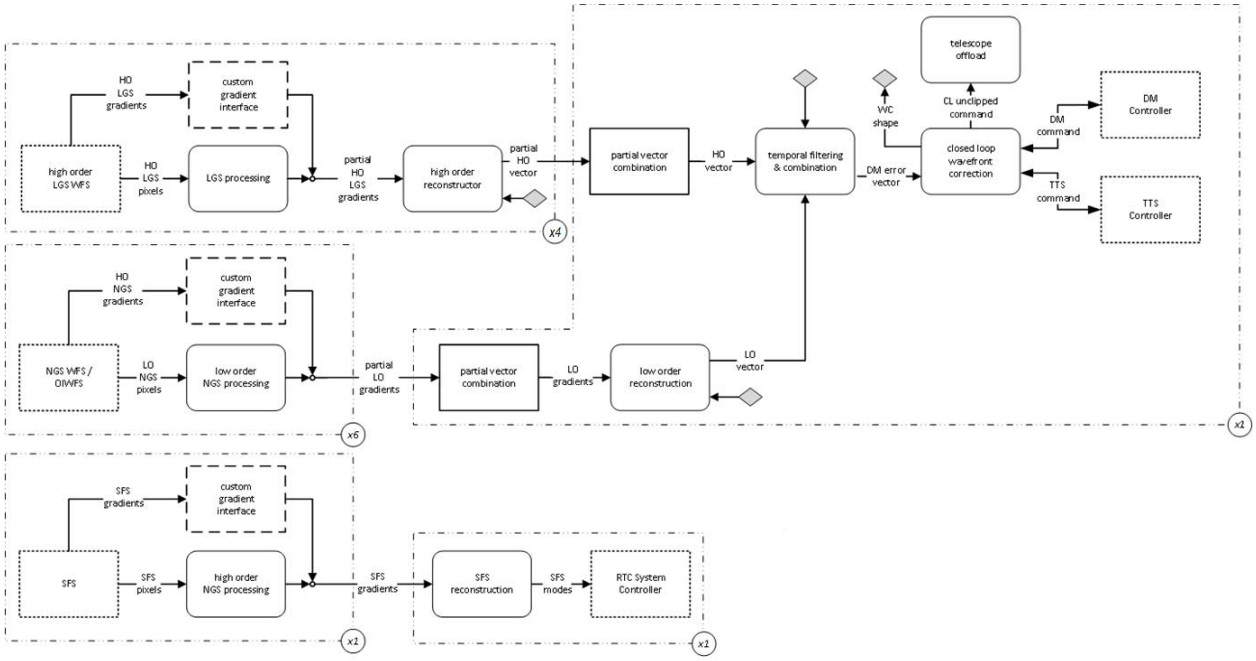


Figure 10-1 – GNAO Template RTC HRT block diagram built with HEART

The GNAO Template RTC HRT pipeline, as shown in Figure 10-1, is nominally modelled as a prototypical LTAO/GLAO system. Four LGS WFS pixel streams are processed ("LGS processing" box in Figure 10-1) and reconstructed in parallel then summed and filtered ("high order reconstructor") to produce a total High Order (HO) vector ("partial vector combination"). Up to six NGS WFS pixel streams, are also processed in parallel ("low order NGS processing") and the aggregated gradients are used to reconstruct a Low Order (LO) mode vector ("low order reconstruction"). The low order modes are filtered and projected into DM command space to be summed with the high order error vector ("temporal filtering & combination"). The net error vector is integrated and then decomposed into a virtual DM and further decomposed into DM commands and tip/tilt commands. These commands are filtered and resampled for telescope offloading ("telescope offload"). Slow focus sensor (SFS) pixels or gradients are processed ("high order NGS processing") to compute focus, and other modes, to be offloaded ("SFS reconstruction").

10.1 CONFIGURATION FILE

Details on how the configuration file is read is covered in the HEART document [RD1]. The GNAO RTC configuration parameters are shown in Table 10-1. Details of what blocks use the configuration information is found in the Internal ICD [RD14]. There is only one configuration file for a single RTC that each block absorbs the relevant configuration information from.

Upon startup of the RTC, configuration information is loaded from a file, and any files that are referenced are loaded. All files are kept under revision control along with the source code. For the GNAO Template RTC, the custom configuration parameters are listed in Table 10-1.

Table 10-1 - GNAO Custom RTC Configuration Parameters

Keyword	Description of the value
Custom for Gemini:	
SFS_WFS_SIZE	number of HO sub-apertures per WFS, and number of slopes
SFS_NUM_MODES	Number of Modes

The standard HEART configuration parameters are described in Section 3.1 of the HEART Internal Interface Document [RD14].

During build and integration there will also likely be experimentation with the core and node assignments for worker threads and CBs to optimize performance for the specific final configuration of GNAO.

The configuration that is specific to the GNAO Template RTC ICD is shown in Table 9-2.

Table 10-2 - GNAO Template RTC Specifications

Specification	Value
number of LGS WFSs	4
number of NGS WFSs	Up to 6
Number of Modes	10
Number of HO sub-apertures per WFS, and number of slopes	20x20, ~2432
Number of LO sub-apertures per WFS, and number of slopes	1x1
Number of LO Modes	6
Number of DMs	1
Number of actuators per DM	393

10.2 STANDARD INPUT & OUTPUT BLOCKS

The Standard Input and Output block will be implemented as discussed above in Sections 9.2 and 9.3. The custom portions of the Standard Input and Output blocks will be designed and implemented after the GNAO bench hardware has been selected.

10.2.1 WFS Interfaces

Custom **wfsReader()** handler functions will be implemented for the 4 x LGS WFS (hrtHoPipe), up to 6 NGS WFS (hrtLoPipe), and the SFS (hrtSfsPipe). Depending on the details of the external hardware connections, the configuration file will likely be augmented to contain connection details (e.g., IP addresses and ports).

10.2.2 WFC Interfaces

Custom **dmDriver()** and **ttDriver()** handler functions will be required for the DM and TTS, and also the **telOffloadDriver**() handler for the secondary control system (all in hrtTfcAndWfcPipe). Also custom are the **lgsFsmDriver()** for the LGS centering errors for the Laser Fast Steering mirrors (i.e. jitter mirrors). Again, configuration fields will be added to store connection details.

11. HARDWARE AND INTERFACES

HEART is designed for a large and demanding AO system, with many measurements (i.e. WFS pixels) and many degrees of freedom (i.e. DM actuator commands) that are, when required, spread over several different physical servers. Most of the AO systems that currently exist at Gemini do not require multiple servers.

HEART will be used to develop the following RTC systems. Details of these systems are provided in the following subsections.

The Template RTC will have interfaces with the following simulated hardware:

- WFSs (5 HO LGS WFSs, up to 6 LO NGS, 1 SFS)
- DMs (3)
- TTS (1)
- SCS (Secondary Control System, this is unique for Gemini)
- LGS Fast Steering Mirrors (FSMs) or Jitter Mirrors
- The System Controller interface.

The GNAO Template RTC will have the following real interfaces:

- WFSs (4 HO LGS WFSs, up to 6 LO NGS, 1 SFS)
- DM (1)
- TTS (1)
- SCS (this is unique for Gemini)
- LGS Fast Steering Mirrors (FSM) or Jitter Mirror
- The System Controller interface.

11.1 GNAO PROPOSED HARDWARE

The proposed GNAO computer hardware, (Table 11-1) consists of a single 2U server with dual AMD 7702 processors. These processors provide enough CPU cores and L3 cache memory to allow efficient wavefront reconstruction via the large matrix vector multiply required for an MCAO system with six high resolution (GNAO 40x40 sub-apertures) LGS WFS and ~3560 DM actuators. This high resolution MCAO system was used to drive hardware requirements and [RD9] provides a detailed comparison between system performance requirements and expected hardware performance. The number of sub-apertures has been reduced to 22x22 with ~1108 DM actuators. The hardware in Table 11-1 was specified for the larger system but will be used as the baseline

server. When it comes time to purchase the hardware, the selection will be revisited to include considering the most recent and best fitting computer hardware.

Table 11-1 - Proposed GNAO computer hardware

Quantity	Description (upgrades / substitutions)
1	Base server 2U chassis, motherboard, redundant power supply, 2 x 1GBASE-T Ethernet
2	AMD EPYC 7702 processor 64 core, 2.0 GHz, 256 MiB cache, 200 W
16	16 GiB DDR4-3200 ECC RDIMM (256 GiB total)
2	256 GB SSD (boot disk)
10	7.68 TB Intel SATA SSD 46 TB storage capacity, 2 parity disks, 2 hot swap spares
1	Hardware RAID controller with 2GB cache & battery backup
5	Intel dual port 10Gb Ethernet (SFP+ fiber ports) **Transceivers and/or cables not included!**

The above server was configured using an online tool at Thinkmate. The tool produced a server cost of $35,494 USD (February 12, 2021) but it should be noted that the SSD telemetry storage accounts for ~$12K USD of the cost and replacing the 64 core 7702 processors with 32 core 7452 processors would save ~$11K USD.

https://www.thinkmate.com/system/rax-qs12-22e2

Table 11-2 AMD EPYC 7002 Series CPU Options

Model	Cores	Base Freq. (GHz)	TDP (Watts)	L3 Cache (MB)	Price ($US)
7702	64	2.00	200	256	Included in configured price
7552	48	2.20	200	192	(-$6,000)
7532	32	2.40	200	256	(-$7,800)
7502	32	2.50	180	128	(-$9,600)
7452	32	2.35	155	128	(-$11,000)

Multiple Ethernet ports are used to ensure that independent traffic streams do not increase system latency by creating temporary bottlenecks. The five LGS WFS controllers send their pixel streams via five dedicated pairs of optical fibers to six dedicated SFP+ ports. Additional dedicated SFP+ ports are assigned the following roles:

- Read NGS pixel streams
- Send DM commands and T/T commands.
- Provide dedicated connection to the GNAO System Controller.

- Provide –streamed telemetry data to external systems.

The tenth SFP+ port is available to provide high bandwidth access to the stored telemetry files. Ethernet interfaces are discussed further in Section 11.5.

The included RAID controller and SSDs provide a fault tolerant boot drive and telemetry storage. The required telemetry storage capacity is estimated at ~35TB without considering the impact of meta-data or filesystem overhead (details can be found in Section 11.4). The provided 46TB usable storage can easily be increased if required by adding additional SSDs to the telemetry RAID array. Each additional SSD would directly increase both the performance and the available storage since additional parity drives and hot spares would not be required.

Since all input and output to the RTC hardware is via fiber-based Ethernet, the RTC server can reside in the Gemini computer room if low latency 10Gb communication to the telescope is available. SFP+ direct attach copper cables can be used to connect the RTC to nearby machines (e.g. within the same or an adjacent rack).

11.2 TEMPLATE RTC HARDWARE REQUIREMENTS

The Template RTC will have interfaces with the following simulated hardware:

- Five LGS WFSs, 1 kHz loop rate, 22x22 subapertures, 220x220 pixels.
- 304 active subapertures.
- Six NGS WFSs, 1 kHz, 1 subaperture, 12x12 pixels.
- Three DMs, ~1108 total actuators (diameters 23, 21, 21 actuators), 1 kHz.
- One TTS, 1 kHz.
- Secondary Control System (SCS); this is unique for Gemini.
- Slow Focus Sensor, up to 10 Hz, 5x5 subapertures, 60x60 pixels.
- LGS Fast Steering Mirrors (FSM) or Jitter Mirrors (five mirrors @ 1 kHz)
- The System Controller interface.

The above dimensions are assumed when assessing the Template RTC hardware required to fulfill the following performance requirements:

REQ-8.3.1.2 NGS Processing Latency	The RTC NGS processing loop shall have a maximum latency, measured from the arrival of the last NGS WFS pixel to output of Tip-Tilt position, of 1 millisecond, with a goal of 500 µs.
REQ-8.3.1.3 NGS Processing Jitter	The RTC NGS processing loop shall have a maximum jitter, measured as a standard deviation over 60 seconds, of 100 µs, from arrival of the last pixel to delivery of the tip-tilt command to the actuator.
REQ-8.3.2.2 LGS Processing Latency	The RTC LGS processing loop shall have a maximum latency, measured from the arrival of the last LGS WFS pixel to output of the last DM actuator, of 500 µs.
REQ-8.3.2.3 LGS Processing Jitter	The RTC LGS processing loop shall have a maximum jitter, measured as a standard deviation over 60 seconds, of 100 µs, from arrival of the last pixel to delivery of the DM commands to the actuators.

A detailed discussion on how these performance requirements can be met with the current design can be found in Section 14.2.

HEART allows multiple instances of a block or pipeline to be implemented when multiple related data flows must be processed.

The EPYC 7002 series CPUs include models with high core counts and large amounts of L3 cache. Each EPYC CPU package consists of one I/O die and up to eight CCD (Core Complex Die). Each CCD contains a pair of CCX (Core Complex) which in turn consist of 4 cores and 16 MiB L3 cache. The CPU cores within a CCX share the L3 cache but must use the I/O die, which acts as a central hub for the processor, to communicate with any cores, cache, memory or interfaces outside of the CCX (e.g. Ethernet via PCI-e).

The model 7002 CPU above contains eight CCD, which equates to 16 CCX, 64 cores and 256 MiB L3 cache. Initial GNAO benchmarking (see Section 15.1) used three cores of an older Xeon E5-2699 v3 CPU to read one LGS WFS pixel stream, calibrate the pixels, compute gradients and perform an MVM to convert gradients into DM vectors. For GNAO, it is natural to have each LGS WFS processed in a separate CCX. Although not necessary, these CCX could be assigned to distinct CCD to reduce execution time jitter by not having similar threads within the CCD attempting to access main memory simultaneously (the LGS WFS streams would be quite synchronized). Another CCX could be used for performing the low order pixel reading and reconstruction and another CCX could perform the final close loop wavefront correction.

In the following table, the hardware resources of the baseline server are allocated to the various processing blocks required by the Template RTC. The following sizing assumptions are made:

- Calculations are performed using 32-bit floating point artihmetic
- 220 x 220 pixels per LGS WFS frame (16 bits / pixels : ~100 KB / frame)
- RTC supports accepting each frame of LGS WFS pixels within 250 µsec
- Pixel calibration performed in floating point (~400 KB for flat + dark)
- 1035 active DM actuators, 1108 total physical actuators
- 484 sub-apertures (10x10 pixels),
- 304 valid sub-apertures per LGS WFS
- High order LGS control matrix: 1140 rows (padded) x 608 colunms ~ 2.8 MB / WFS
- Six NGS WFSs, 1 kHz, 1 subaperture, 12x12 pixels.
- Three DMs, ~1108 total actuators (diameters 23, 21, 21 actuators), 1 kHz.
- TTS, 1 kHz.
- Secondary Control System (SCS); this is unique for Gemini.
- Slow Focus Sensor, up to 10 Hz, 5x5 subapertures, 60x60 pixels.

The hardware allocations below are meant to show that the baseline server hardware is more than adequate for the Template RTC. It is likely that performance requirements would be met even with significantly more sharing of hardware resources but that sharing would be determined after more performance measurements are made on the developed software and in concert with hardware selection.

Table 11-3 – Hardware resource allocation to blocks

# CCX	10GbE	Processing Blocks	Resource Availability L3 Cache & Network Bandwidth	Comments, Resource Requirements (Cache memory, bandwidth)
5	5	LGS processing, HO Reconstruction	Cache/WFS: 16 MiB L3 Ethernet/WFS: 10 Gb/sec	Per LGS WFS: 200 KB (raw pixels) 1 MB (calibrated pixels & 2 x calibration coefficients) 2 x 2.8 MB control matrix ~7 MB : cache subtotal 3.7 Gb/s Ethernet (raw pixels + 20% overhead in 250 µsec)
1	1	NGS processing, LO partial vector combination, LO Reconstruction SFS processing & reconstruction	16 MiB L3 10 Gb/sec Ethernet	~30 KB total (raw & calibrated NGS & SFS pixels, calibration coefficients, reconstruction matrices, etc.) 0.3 Gb/sec (accept SFS pixels & all NGS pixels within 250 µsec)
1	1	High order partial vector combination, Temporal Filtering & Combination, Closed Loop Wavefront Correction		Cache: Ethernet: Low bandwidth but independent link eliminates increased latency due to unrelated network traffic. Send T/T commands & DM vector
1	1	Store telemetry data to disk & stream live (decimated) telemetry data to clients	16 MiB L3 10 Gb/sec Ethernet	Telemetry storage bandwidth (Section 11.4) is less than 10% of expected bandwidth from RAID array (> 2 GB/s). Streaming to external clients can be throttled but one 10Gb/s link is higher bandwidth than required for non-decimated telemetry data.
1	2	Command/Data Interface to System Controller NFS mount of RTC telemetry data	16 MiB L3 cache 2x10 Gb/sec	Commands & data between HRTC & System Controller Ethernet: One link provides communication between RTC and system controller.

				Second link provides dedicated link to telemetry storage data so that downloading files does not impact system controller commands or RTC loop operation.
1	0	RPG handler	16 MiB L3 cache	Relaxed latency requirements
10	10	Total resource allocation		
6	0	Unallocated (available) resources		
16	10	Total hardware resources		

11.3 GNAO RTC HARDWARE REQUIREMENTS

The GNAO Template RTC will have the following real interfaces:

- Four LGS WFS, 1 kHz loop rate, 22x22 subapertures, 220x220 pixels.
- Six NGS WFS, 1 kHz, 1 subaperture, 12x12 pixels.
- One DM, ~416 actuators, 23 actuator diameter, 1 kHz.
- TTS, 1 kHz.
- Secondary Control System (SCS); this is unique for Gemini.
- Slow Focus Sensor, up to 10 Hz, 5x5 subapertures, 60x60 pixels.
- LGS Fast Steering Mirrors (FSM) or Jitter Mirrors (4 mirrors @ 1 kHz)
- The System Controller interface

Since the GNAO RTC contains significantly fewer DM actuators and fewer LGS WFS, the fact that the Gemini Template RTC will meet the Gemini performance (latency and timing jitter) requirements implies that the GNAO RTC will also meet those same performance requirements.

11.4 TEMPLATE RTC TELEMETRY STORAGE REQUIREMENTS

In order to verify that the proposed hardware will meet the telemetry storage requirements, the following data retention periods are assumed.

- Raw WFS pixels stored at 50 Hz (1 day)
- Slope calculations at full frame rate (30 days)
- Control matrices (8 days)
- Intermediate data (8 days)
- DM & Tip/Tilt commands (30 days)
- Corrections sent to external systems (8 days)

Note that based on requirement 8.3.4.1, the raw WFS pixels are stored decimated at 50 Hz. Note that it is possible for the pixel to be stored at the full frame rate, if desired. However, the storage drives have been sized to only store one of night of pixels at 50 Hz. Therefore, if pixels are stored at a higher rate, or full rate, then the full night's worth of pixels cannot be stored. As an example, if pixels are stored at 1 kHz, then only ~1/20 of a night's worth of pixels can be stored.

Table 11-4 below summarizes the template RTC telemetry storage requirements. Raw LGS WFS pixels are stored at a decimated rate for one night only. LGS WFS gradients and applied DM

actuator values are stored at the full 1000 Hz frame rate for 30 days and intermediate data is stored at full frame rate for 8 days. A total of ~32.5 TB usable storage capacity is required.

Table 11-4 Template Telemetry Storage Requirements (Summary)

Storage (TB)	Bandwidth (MB/s)	Retention (days)	Data stream
0.52	24.2	1	Raw LGS WFS pixels (50 Hz)
9.85	15.2	30	LGS WFS gradients
2.87	4.4	30	DM actuator values (sent to DM)
7.88	45.6	8	Intermediate LGS WFS gradients
6.13	35.5	8	Intermediate DM vectors
0.29	1.7	8	High Order Control Matrices
4.99	28.9	8	Other (e.g. OIWFS, TT, LGS FSM, LGS SA flux, etc.)
32.5	**155**	**N/A**	**Total**

The Gemini Template RTC requires ~32.5 TB usable disk capacity for storing telemetry data for the retention periods specified in Table 11-4. Benchmarking done for the NFIRAOS RTC using an 8 SSD RAID-6 array (six data disks, two parity disks, no hot spare) demonstrated streaming data to the array at a rate of ~1800 MB/s using a single program thread. Using three writer threads allowed a total bandwidth of ~2800 MB/sec which is close to the maximum theoretical bandwidth of the six data drives (6 * 485 MB/s = 2910 MB/s).

Even though relatively slow SATA SSDs were used in the benchmark, the bandwidth required by the Gemini Template RTC is ~155 MB/s is much less than the bandwidth achieved from the SSD based RAID array. The required disk write bandwidth is easily achieved with a modern RAID array and array capacity can be easily calculated and adjusted to meet requirements. As higher capacity drives become available, the number of drives used in the RAID array can be reduced since even a single SATA SSD provides significantly more bandwidth than the required 155 MB/s.

The proposed hardware includes ten 7.68 TB SSDs, six provide usable data capacity, two are used for error correction (i.e. RAID-6) and two are hot spares. The RAID-6 array could be reduced to a total of eight SSD which would provide 5 * 7.68 TB ~ 38 TB usable storage and a single hot spare.

11.5 INTERFACES WITH THE ETHERNET HARDWARE

HEART has been developed around the concept of using Ethernet for communication with both wavefront sensors and wavefront correctors. Using Ethernet-based devices when possible provides several benefits compared to the more traditional approach of using PCI cards to communicate with external devices.

- PCI based hardware device drivers are often only available for particular Linux distributions and/or Linux kernel versions. These version restrictions can become particularly problematic as the number of independent hardware devices within a single machine increases. The use of Ethernet as the method of communication with the hardware typically removes any requirements for particular Linux versions.
- The use of Ethernet interfaces allows the use of a test server consisting of a Linux server with multiple Ethernet ports to mimic multiple external devices. This allows more realistic testing of the RTC since the test server can stream pixels to the RTC and read

the resulting wavefront corrector commands sent by the RTC. This allows accurate RTC performance measurements and allows testing the RTC without physical hardware and does not require the RTC system to be running the mock devices.

- The use of Ethernet interfaces also eases the testing of the wavefront sensors and wavefront correctors since special cards do not need to be installed in a machine in order to exercise the hardware.

In HEART, streams of pixels or centroids are received from various wavefront sensors. From the RTC's point of view these pixels/centroids are read from a socket and stored in circular buffers within the RTC memory. Any existing WFS and any future WFS simply requires a thin interface layer to translate the Ethernet data stream to the internal HEART data structure. This means that the same architecture can be used regardless of the type of WFS. The same is true of the interface for the Deformable Mirror electronics and tip/tilt stage. It should be noted that with modern Ethernet and careful configuration of the links, the additional latency caused by the Ethernet transmission can be made quite low (e.g. 10-20 microseconds or less). This feature will be valuable for future upgrades.

It is important that the input pixel (or gradient) streams and the output wavefront corrector streams are not subject to network congestion. Table 11-5 demonstrates how the RTC Ethernet data streams use different physical links to reduce traffic contention and timing jitter. Increased traffic segregation results in lower latency, more predictability and better control of dropped frames due to buffer overflows.

Table 11-5 - Gemini Template RTC Server Ethernet Ports

Quantity	Server Ports	Data Streams	Notes
5	5	LGS WFS Pixels	Independent 10Gb Ethernet link for each LGS WFS data stream to reduce latency and timing jitter.
1	1	NGS WFS pixels/gradients	Up to six small data streams over a single Ethernet port.
0	0	SFS pixels/gradients	Uses NGS Ethernet port
3	1	DM Controllers	A single Ethernet port can be used to send DM commands to all three deformable mirrors since the 10Gb Ethernet bit transfer time for 1108 floating point actuator values is less than 4 µs.
1	0	TTS	Shares DM port.
1	0	LGS FSM (LGS jitter control)	Shares DM port
1	1	System Controller	Dedicated Ethernet port for communication between RTC and System Controller.
1	1	External Telemetry Sink	Streaming telemetry data should impact neither the real-time data flows nor the RTC / system controller command channel.
1	1	System Controller (NFS)	Optional separate Ethernet link between System Controller and

			RTC used to provide NFS access to stored telemetry data. Moving NFS traffic to a separate link reduces communication latency with the System Controller.
	10	Total server ports	

By segregating the RTC data streams appropriately and configuring the RTC system for low latency network traffic, a multi-core Linux server with multiple low latency Ethernet links can provide the hardware for a low latency HEART based RTC.

12. OBSERVATORY INTERFACE

The interface mechanism with the observatory is unique for Gemini but the communication protocol and some of the commands and status are not. The control of the RTC is hierarchical, with the System Controller orchestrating the commands and configuration required to control the RTC (Figure 12-1). Experience has taught us that having a highly-configurable AO system, particularly including the ability to change the sequence of events that leads to closing the AO loops, is essential when commissioning a new instrument. To this end, the RTC will provide an interface with different low-level commands, and therefore highly fine-grained control, made available to the System Controller. This ability gives Gemini abundant flexibility for future AO instruments.

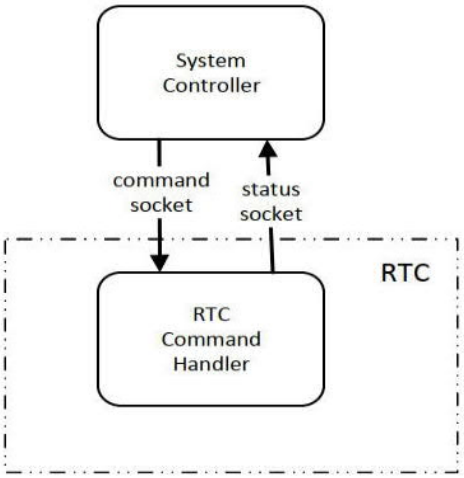


Figure 12-1 - Command Handler to System Controller

There is a command socket between the System Controller and the RTC Command Handler that handles all of the command interactions. There is a subset of commands that are common for all RTC's and are listed in [RD1]. There is a subset of commands that are specifically for the Template and GNAO Template RTC. Those are listed in the custom external interface document [RD13].

There are status/state/telemetry passed back to the System Controller that are sent using a separate status socket. To avoid having to use all of those names, it will just be called status. This can include:

- status of a command (this is used for testing purposes and is not to be confused with the command acknowledgment and command completion on the command socket)
- state of the various loops
- LGS Focus errors
- focus and coma corrections
- NGS centering errors for the NGS WFS controller
- field rotation offsets (when using >=3 NGS)
- primary mirror figure error corrections

13. OPERATIONS

This section describes the functionality that the AO instrument and RTC will need to perform at different times. These sequences give a clear view of the operation of the RTC and shows that the required functionality is supported. It is recommended that this section be extracted into a standalone document that shows what types of commands are passed to perform specific sequences to help software engineers understand what needs to be supported in order for the AO instrument and RTC to work together. This should be led by the System Controller architect.

The operations are divided into five categories: Startup, Calibrations, Observation, Shutdown, and Engineering tasks, as depicted in Figure 13-1 and discussed in the following sections.

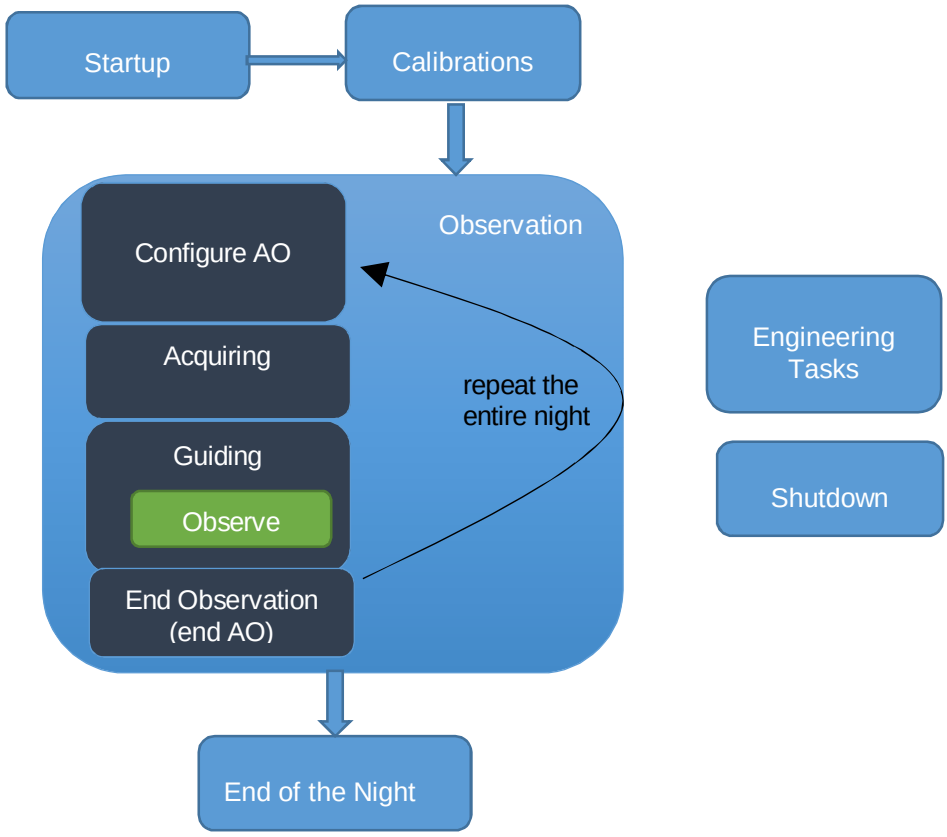


Figure 13-1 - AO Sequencing

13.1 STARTUP (BEGINNING OF THE NIGHT)

These are the sequences that are executed once each night to prepare the AO instrument and include everything that is needed to be done to get the instrument from its current state to where an observation can be started. It is expected that computers will remain on once installed and configured. Startup at the beginning of the night includes:

- restore the RTC software to the state immediately after software startup. This will cause all configuration files to be (re)read and reset any parameters. It will also deactivate the pipeline, reset all internal states to default values and clear any mode configuration parameters. The RTC stores telemetry data over a specified time frame, and, at the beginning of the night (or when it is scheduled), and when aged files are removed.

This sequence will prepare for day/night time calibration and night time observations. This sequence may be from a cold start (powered off) or perhaps from a state where the system was parked from the previous evening. In late afternoon, all the components in the AO instrument will go through initialization and find their known datum positions that may include self-tests.

13.2 CALIBRATIONS

Ideally, this includes any calibration that does not require a dark sky, as night operations are more costly. This can include calibrations that need to be done daily with a short time frame or anything that can be done during the day using simulated light sources. Daytime or sunset calibrations include sequences that are done frequently or at least can be more predictably scheduled. This is different from tasks that occur during commissioning or that occur infrequently (Engineering Tasks). These are tasks such as taking detector dark frames.

Note that while these calibrations will generally occur after startup, and before observations, nothing precludes these activities from occurring during the night.

13.3 OBSERVATION

An observation can include multiple exposures and multiple targets, and is predefined from an observing recipe executed by the OCS. The AO instrument will be configuring while the Telescope is slewing and then will be actively correcting prior to the start of the science observation. During an observation, the AO instrument may have to track guide stars with the loops closed if doing non-sidereal tracking.

This phase will include configuration of the AO instrument, the RTC, acquiring the guide stars, then, when steady state is reached with the AO loops observations, proceed to saving telemetry data as it is acquired. This time period is further subdivided into the following:

- **Configure AO**: The System Controller will configure the AO instrument and the RTC with new configurations/targets. This occurs while the telescope is slewing. This will select the RTC mode, which will:
    - Stop any running pipeline
    - Configure the SRT – RPG so that initial control parameters can be created for the RTC
    - Configure the High Order WFSs
    - Configure the Low Order WFS(s)
    - Assign Loop rates
    - Configure wavefront correctors
    - Once completed, it is assumed that the triggers are started on the WFS to start data collection.

  Once the AO instrument is configured, it then starts all the assemblies that require tracking of the telescope, eg ADC's, etc.

  When done, the RTC pipeline will be activated. It will:

    - Flatten DMs,
    - Zero TTS, zero FSM
    - Reset Integrators and control parameters
    - Start reading pixels or gradients
        - If receiving pixels then gradient computation will be started
        - At this time there may be no light on the WFS's, and that is acceptable, the GUI's could start streaming in preparation

- **Acquisition:** This occurs when the telescope has slewed into position and there is light on the guide stars. Guide stars are acquired, which may involve many different steps. The following is an example of a possible sequence (but there will also be alternate acquisition sequences depending on the guide stars):
    - Enable sending of the LGS FSM commands to stabilize the LGS spots
    - Close the LGS HO loop
    - Start HO Optimization Processes
    - Close the LO Loop (if configured)
    - Start SFS processing path
    - Allow the system to reach steady state
    - Start all Optimization Processes

- **Observe**: When the AO loops are closed then observation block(s) are executed. This can include a dither where the guide star light stays within the field of view of the AO system during an observation.

- **End AO**: If the telescope is going to be slewed, or the RTC needs to be re-configured, then the Guiding will stop and the loops will open.

There will be different variations to this scenario that should be captured in the sequences document. Some examples include:

- Loss of lock on a guide star
- Re-acquiring a single guide star
- Non-sidereal tracking

13.4 END OF THE NIGHT

End of Night sequences constitute everything required for the AO instrument and RTC at the end of each observing night. From the RTC point of view, this could include tasks such as making sure no guiding is being performed, putting the DMs to the system flat position, zeroing the tip/tilt stage, and performing anything that the AO instrument requires such as de-energizing the DM electronics, or parking all mechanisms.

13.5 ENGINEERING TASKS

These are sequences that are done infrequently, for example generating flats for the RTC, pointing models, or taking images for alignment for the AO instrument. These also include sequences to be done during commissioning or as a means to debug functionality.

13.6 SHUTDOWN

The shutdown sequences relate to preparing the GNAO instrument and RTC software to be shut down. This may include activities such as powering down components, flattening DMs, zeroing tip/tilt, etc. This is based on the needs of the instrument and would be commanded by the System Controller.

14. PERFORMANCE

HEART can be used to implement many different types of AO systems and a comparison between those systems are discussion in the HEART Design Description Document [RD1]. For these comparisons note that:

- Sizes of control matrices are given in MiB (1 MiB = 1024 * 1024 bytes).
- Bandwidth values are given in GB (1 GB = 1,000,000,000 bytes).

CPUs required are based on 8-channel DDR4-3200 memory (204.8 GB/s per CPU). For reference, an AMD 7xx2 CPUs includes 8 channels of DDR4-3200 memory (130-150 GB/s per CPU using industry standard STREAM benchmark). Achievable bandwidth is dependent upon code and system configuration.

14.1 PERFORMANCE CONSIDERATIONS

Consider a large AO system with N_{Act} actively controlled (non-extrapolated) actuators, N_{SA} illuminated subapertures, and N_{Pix} WFS pixels. If an MVM based reconstructor is used, the $O(N_{Act}*N_{SA}+N_{Act}+N_{SA}+N_{Pix})$ computational requirements and memory bandwidth of the system are driven by the $O(N_{Act}*N_{SA})$ size of the control matrix.

In a HEART based system, the control matrix is partitioned to allow parallel computation. This parallel MVM computation, along with the other processing threads are explicitly assigned to computing resources (e.g. CPU cores) to reduce system latency.

For GNAO, the assumed DM actuator counts are given in the table below. The assumed DM diameters, in actuators, are given in the table below. For each DM in the table, the number of actuators per quadrant is computed and rounded up to an integer number of actuators. This is then scaled to the full actuator count. This provides a slightly larger actuator count than simply using π Diameter^2 / 4. The assumed DM diameters are based on an LGS WFS diameter of 22 subapertures.

Table 14-1 – GNAO DM Dimensions

DM	Diameter	Act / DM
DM0	23	416
DM5	21	348
DM14	21	348
Total	N/A	1112

High order MVM is not the bottleneck if it can be performed within 300 µs.

The current RTC baseline uses dual socket AMD EPYC 7002 series servers; each CPU supports eight channels of DDR4-3200 memory, providing a theoretical memory bandwidth of 8 DIMMs * 8 bytes/transfer * 3200 MT/s (megatransfers/s) ~ 200 GB/s. Industry standard STREAM benchmarks indicate that ~150 GB/s or more can be achieved in practice. An EPYC 7xx2 CPU can contain up to 64 cores and 256 MiB L3 cache.

Table 14-2 below shows the data sizes and computational requirements for NFIRAOS and several Gemini systems. The table only addresses the computational requirements of the high order MVM and is useful as a rough tool for sizing the required computer hardware.

The 300 µs used as the required MVM execution times for GNAO and GPI 2.0 is assumed to be less than the LGS WFS readout times; this allows the RTC to compute the MVM as quickly as

possible as new gradient values become available. As an example LGS WFS, the 240x240 pixel OCAM2K detector uses 32 outputs at 5Mpix/sec for a theoretical readout time of ~360 µs.

Table 14-2 – Data Sizes and Computational Requirements for various AO systems

	NFIRAOS	Template RTC for Gemini	GEMS	GPI 2.0
Loop Rate (Hz)	800	1000	800	2000
# Subap across WFS	60	22	16	60
#Active DM Act. (CM rows)	6981	1035	684	1521 (π 22^2)
# WFS	6	5	5	1
Pixels / LGS WFS	204,792	48,400	6,400	32,400 (180 x 180)
# Subap / LGS WFS	2896	484	256	2827 (π 30^2)
#Subap / LGS WFS (illuminated)	2700	304	204	2827 (π 30^2)
# Gradient values / LGS WFS	5400	607	408	11,308 (pixels)
Total Control Matrix Size (MB)	905	12.6	5.6	68.8 + 9.25
Required MVM Execution Time (µs)	500	300	500	300
Required Bandwidth (GB/s)	1810	42	11	N/A (read from L3 cache)
# CPUs Required (8 x DDR4-3200)	8.8	0.3	0.1	
# MVM Servers x #CPU / server	6 x 2	1 x 2	N/A	1 x 2

From the table above, it is clear that none of the Gemini AO systems listed will require multiple RTC servers. In addition, the control matrix for each Gemini system is small enough to be double buffered within the L3 cache of a single CPU. This will allow loading of the control matrix from RAM and switching to a new control matrix after it has been fetched into the L3 cache.

GPI 2.0 is the most computationally demanding Gemini system in the table. In GPI 2.0, in addition to the 68.8 MB control matrix, a 9.25 MB modes to actuators transformation matrix is also required. . This is because GPI 2.0 uses modal control. The GPI 2.0 RTC does not need to fetch new matrices during the MVM calculations and both MVMs are executed with coefficients stored in L3 cache.

To estimate the Template RTC computational requirements, initial estimates using isolated benchmarks and extrapolation from tests of the NFIRAOS RTC design were performed. Additionally, the NFIRAOS RTC benchmark code was reworked to demonstrate the viability of a single-server approach for the GNAO RTC. These latest results are shown below in Section 14.1.1 while the earlier estimates are provided in Section 14.1.2.

14.1.1 Updated GNAO RTC Benchmarking

For the purposes of supplementing our initial timing estimates for Template RTC, we have reworked tests originally developed to guide the multi-server NFIRAOS RTC design. We simulate

the full GNAO RTC using just one of the same servers that were purchased for NFIRAOS benchmarking (now several years old), but modify the setup of our software to accommodate the dimensions of GNAO, including 608 gradient values for each of the five LGS WFS, and 1135 active DM actuators, running at 1000 Hz.

Pixel streams for five LGS WFSs are generated by a single application called rtcPixels on one test server, and sent over 5 x UDP sockets (one per LGS WFS), using two 10 GbE ports (three streams on one port, and two on the other), throttled so that the pixels are spread ~evenly over a requested period, to an application on a second test server called rtcBenchmark (which also has two corresponding 10 GbE ports). This second application receives the pixels, calibrates them, measures gradients, performs the partial MVMs for each LGS WFS, and combines the results to produce DM vectors. For each frame the resulting DM vector is sent by rtcBenchmark back to the rtcPixels application on the first test server over a single UDP socket so that end-to-end timings can be measured.

The first test machine, nrtc-wcc (WFS pixel simulation and round-trip timing), is a dual-socket server with 2 x 3.4GHz Intel Xeon E5-2643 v4 CPUs (released in 2016). The second machine, nrtc10 (RTC simulation), is also dual-socket, with 2 x 2.30 GHz Intel Xeon E5-2699 v3 CPUs (released in 2014).

The nrtc10 server was chosen to benchmark the GNAO RTC since a single CPU has 18 cores, enabling efficient multi-threading of the pixel processing and MVMs, and a 46 MB cache which can easily hold the control matrices for all five LGS WFSs. The second CPU is thus free to execute a Python program simulating the SRT System computing a reconstructor as part of the RPG (details provided below). The system is further configured in the following ways:

- a real-time patched kernel is used;
- the OS and regular applications are restricted to the CPU0 (cores 0—17);
- the numactl command is used to run rtcBenchmark restricted to the CPU1 (cores 18—35), and uses only the memory attached to that CPU; and
- the Ethernet interrupts for the two 10 GbE ports are assigned to cores on CPU1 so that all network traffic goes directly to the CPU running the RTC.

The cores of CPU1 on nrtc10 are partitioned for the RTC benchmark software in the following way:

- 18—22: one thread to read pixels for each LGS WFS
- 23—27: one thread to calibrate pixels and calculate gradients for each LGS WFS
- 28—32: one thread to calculate the partial MVM for each LGS WFS
- 33: the main thread
- 34—35: one thread to handle Ethernet interrupts for each of the two adapters
- The cores on CPU0 run the Python program computing the reconstructor as part of the RPG.

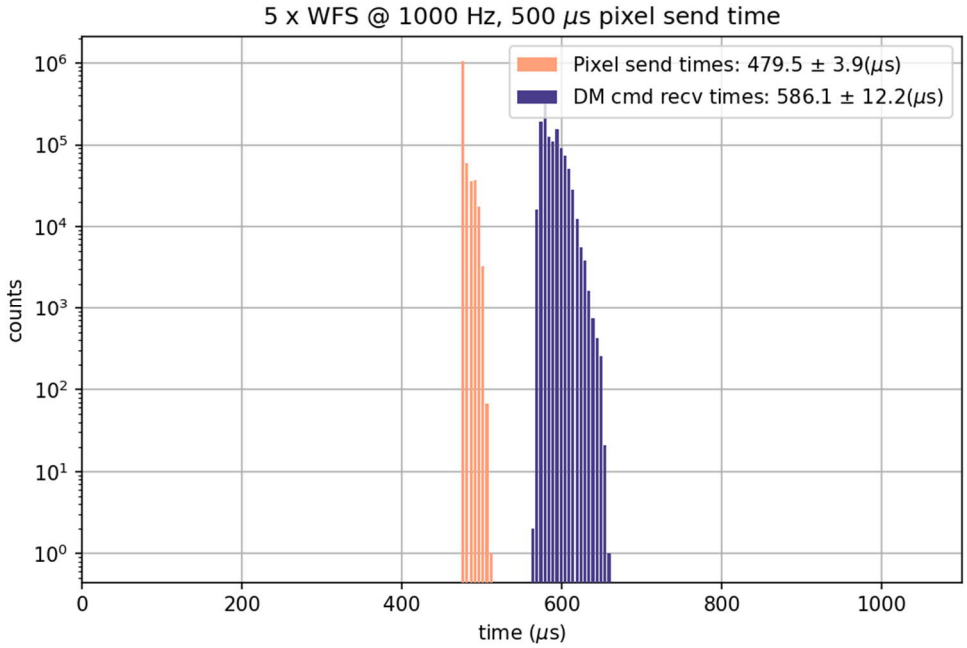


Figure 14-1 Round trip timing histograms for baseline 5 WFS system at 1000 Hz

Figure 14-1 shows the overall performance of the system during a 20-min run as measured with a single clock on rtc-wcc. The x-axis (time) zero-point corresponds to the start of a new frame of LGS WFS data at which point rtcPixels begins sending pixel data as UDP datagrams (22 datagrams per WFS per frame). The pink “pixel send times” histogram corresponds to the last datagram being sent; specifically, once the socket send() function has returned for that datagram. The RTC benchmark program then calculates a DM vector and sends it back to rtcPixels so that it can measure the time for delivery using the same clock, whose distribution is shown by the blue “DM cmd recv times” histogram. This measurement approximates the RTC latency from pixel reception to DM command delivery. This plot shows that this latency is no more than 106.6 µs (= 586.1 – 479.5) after the sending of the last pixel on average (noting that this is slightly pessimistic since the actual pixel arrival time at the RTC is slightly later than the send time used here). It also shows that the RMS jitter is no worse than 12.2 µs (since the round-trip times include the small ~3.9 µs scatter in the pixel send times).

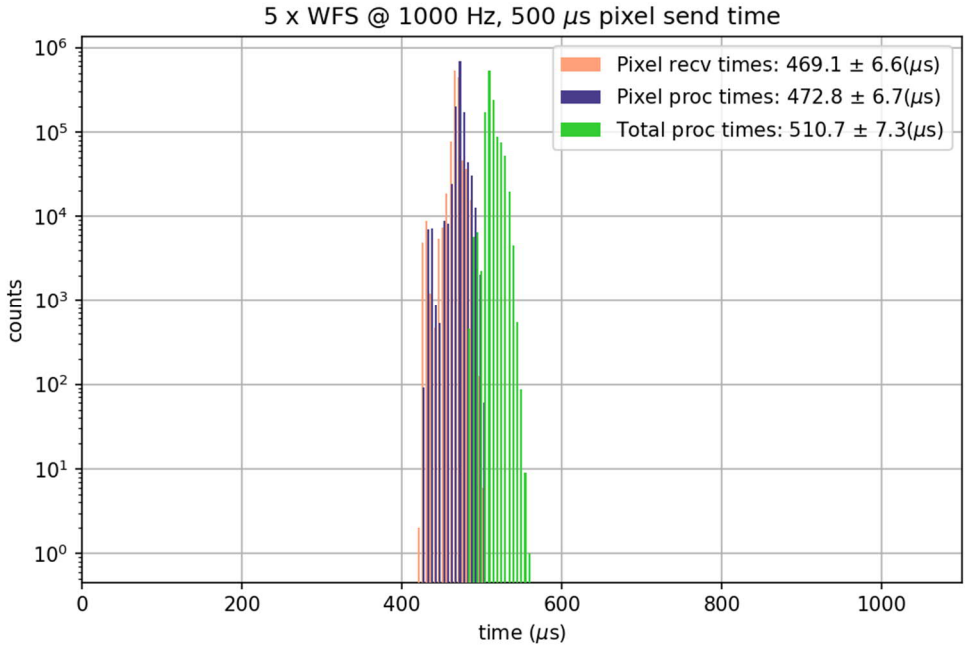


Figure 14-2 RTC execution histograms for baseline 5 WFS system at 1000 Hz

Figure 14-2 shows detailed execution timings from rtcBenchmark. The x-axis (time) zero-point for this plot is established as the time when the first pixel datagram is received from any of the LGS WFS. The timings in this plot are therefore shifted slightly to the left of the previous figure, and there is some scatter due to the WFS -> RTC network latency (i.e., the pink "Pixel recv times" for the RTC do not in fact indicate that pixels are received faster than they are sent!). Regardless, note that: (i) this plot provides accurate relative latencies of processing steps performed within the RTC; and (ii) the round-trip times of the previous plot demonstrate the absolute performance of the system. The first key point is that the "Pixel proc time" histogram (blue, which includes pixel calibration and measurement of gradients) is able to keep up with the "Pixel recv time" histogram (pink, which simply reads pixels from the socket), incurring a negligible 3.7 μs (=472.8 – 469.1) latency, and virtually no additional jitter. The remaining processing, which consists of the partial MVMs (one for each WFS), and combination to produce a final DM vector, and finally sending over a UDP socket only adds an additional ~37.9 μs (=510.7-472.8) latency following the end of pixel processing (the "Total proc times" in green). Note that in our core assignments only a single thread is available for each partial MVM (the calculation that drives the total processing time). The code was originally designed to partition the MVM over several threads, so a modern CPU with more threads, or a faster clock speed per core, could certainly tighten timings up further, though they already easily meet the requirements of GNAO.

Another point to consider is the potential impact on latency and jitter incurred by other processes (particularly the SRT RPG) running on the same computer. Past experience with the NFIRAOS RTC benchmarking effort has demonstrated that: (i) use of an RT-patched kernel; (ii) dedicating an entire CPU of a multi-socket system to the RTC; (iii) giving RTC processes exclusive use of the memory attached to that CPU; and (iv) having the traffic of dedicated NICs directed to that CPU, effectively insulates the RTC processes from activities on the other processors. As a practical demonstration of this point, the above simulation was executed in parallel with a Python script used to prototype the calculation of the MV Reconstructor, as summarized in Section 5.4.3 and with further details in Section 5.3.1.1.4 of the HEART Design Document [RD1]. This Python code is restricted to the CPU0, and uses numerical libraries (e.g., NumPy and SciPy) to perform linear algebra with parallelization over available cores. The following screenshot shows the output of top while the RTC and Reconstructor benchmark codes were running. It has been configured to show individual threads, and the second-to-last column indicates the core. It clearly shows Python running on the lower 18 cores (first CPU), with the RTC benchmark application running on the upper 18 cores.

```
top - 11:26:38 up 4 days,  3:11,  6 users,  load average: 3.49, 6.61, 7.40
Tasks: 657 total,  34 running, 623 sleeping,   0 stopped,   0 zombie
Cpu(s): 17.5%us,  0.9%sy,  0.0%ni, 81.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  131754896k total, 20167300k used, 111587596k free,   196336k buffers
Swap: 10485756k total,        0k used, 10485756k free,  8320440k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+   P COMMAND
 6922 chapine   20   0 11.4g 9.4g  17m R 94.9  7.5  21:15.33  6 python3
 7173 chapine  -25   0  192m 180m 2548 R 52.7  0.1   8:26.75 20 rtcBenchmark
 7174 chapine  -25   0  192m 180m 2548 R 52.7  0.1   8:27.74 21 rtcBenchmark
 7171 chapine  -25   0  192m 180m 2548 R 52.4  0.1   8:27.57 18 rtcBenchmark
 7172 chapine  -25   0  192m 180m 2548 R 52.4  0.1   8:27.32 19 rtcBenchmark
 7175 chapine  -25   0  192m 180m 2548 R 52.1  0.1   8:26.74 22 rtcBenchmark
 7166 chapine  -23   0  192m 180m 2548 R 49.4  0.1   7:57.30 28 rtcBenchmark
 7168 chapine  -23   0  192m 180m 2548 R 49.1  0.1   7:56.19 30 rtcBenchmark
 7167 chapine  -23   0  192m 180m 2548 R 48.8  0.1   7:54.34 29 rtcBenchmark
 7169 chapine  -23   0  192m 180m 2548 R 48.8  0.1   7:54.34 31 rtcBenchmark
 7170 chapine  -23   0  192m 180m 2548 R 48.8  0.1   7:54.79 32 rtcBenchmark
 7177 chapine  -24   0  192m 180m 2548 R 47.8  0.1   7:41.81 24 rtcBenchmark
 7178 chapine  -24   0  192m 180m 2548 R 47.8  0.1   7:40.00 25 rtcBenchmark
 7179 chapine  -24   0  192m 180m 2548 R 47.8  0.1   7:41.09 26 rtcBenchmark
 7180 chapine  -24   0  192m 180m 2548 R 47.8  0.1   7:40.23 27 rtcBenchmark
 7176 chapine  -24   0  192m 180m 2548 R 47.4  0.1   7:41.22 23 rtcBenchmark
 6924 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   4:11.49  8 python3
 6925 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   4:05.07 14 python3
 6931 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.99  3 python3
 6932 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.78 10 python3
 6933 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.90  7 python3
 6934 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.73 11 python3
 6935 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.67  1 python3
 6936 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.35  4 python3
 6938 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.28  5 python3
 6939 chapine   20   0 11.4g 9.4g  17m R 33.2  7.5   3:40.64 16 python3
 6926 chapine   20   0 11.4g 9.4g  17m R 32.8  7.5   3:41.18  9 python3
 6927 chapine   20   0 11.4g 9.4g  17m R 32.8  7.5   3:41.21  0 python3
 6928 chapine   20   0 11.4g 9.4g  17m R 32.8  7.5   3:41.17 17 python3
 6929 chapine   20   0 11.4g 9.4g  17m R 32.8  7.5   3:41.29  2 python3
 6930 chapine   20   0 11.4g 9.4g  17m R 32.8  7.5   3:41.02 12 python3
 6937 chapine   20   0 11.4g 9.4g  17m R 32.8  7.5   3:40.51 15 python3
 6940 chapine   20   0 11.4g 9.4g  17m R 32.8  7.5   3:40.24 13 python3
```

Finally, noting that there is a goal to run GNAO RTC at up to 2 kHz, we ran a second simulation at that rate, with the pixel send time reduced to 250 µs. Due to the inability of the MVM to keep up at this rate, the number of threads assigned to each partial MVM was increased to two. However, the number of cores required would then exceed the number available on the CPU, so for this test the number of LGS WFS was reduced to four. The benchmark test is otherwise identical to the previous test (including running the Python reconstructor in parallel).

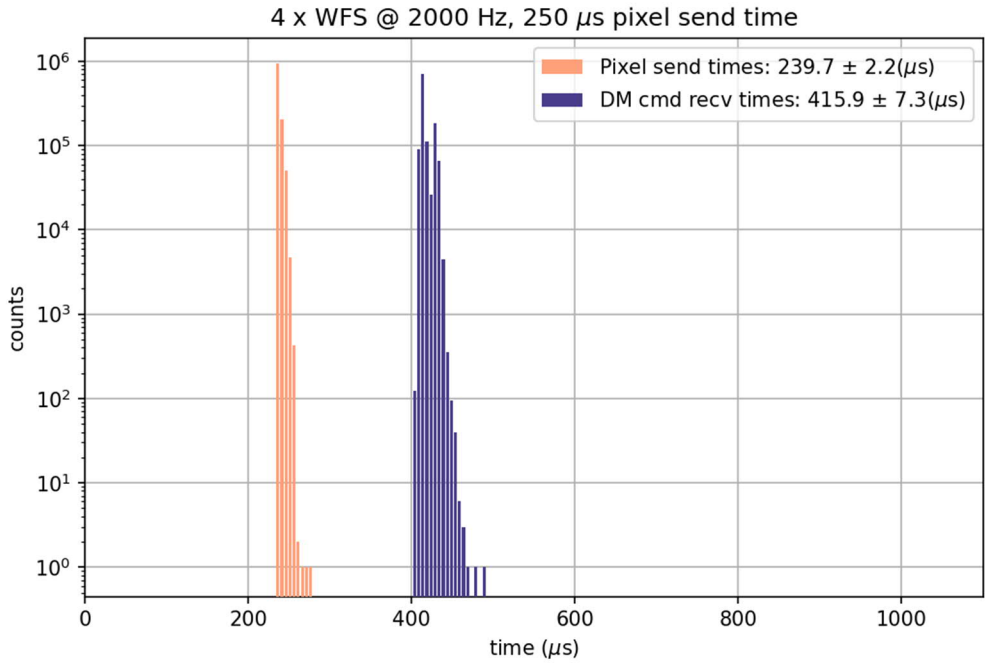


Figure 14-3 Round trip timing histograms for 4 WFS system at 2000 Hz

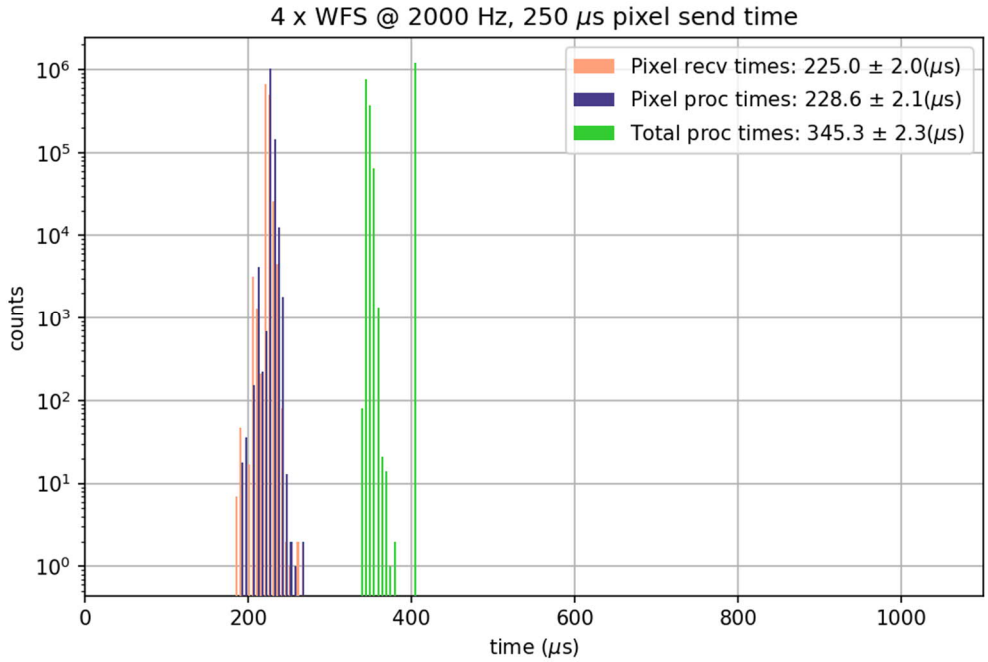


Figure 14-4 RTC execution histograms for 4 WFS system at 2000 Hz

Figure 14-3 shows the pixel sending and round-trip times, while Figure 14-4 shows the RTC execution times, for this modified benchmark running at 2000 Hz. Again, pixel processing only incurs a small latency of 3.6 µs (=228.6 – 225), and the MVM + remaining processing 116.7 µs (=345.3 – 228.6. The round-trip times have DM vectors being delivered on average 415.9 µs after the WFS begins sending pixels (fitting into the required 500 µs). Using a more modern server we are therefore optimistic that the GNAO RTC will be able to run a 2 kHz.

14.1.2 Earlier GNAO RTC timing estimates and isolated benchmarks

This section summarizes isolated benchmarks and extrapolation of results from earlier NFIRAOS RTC tests to GNAO.

Using the Template RTC values of 1112 active DM actuators and 768 gradient values per LGS WFS, a single threaded MVM was benchmarked using a control matrix with 1112 rows and 768 columns. Figure 14-5 shows the execution time histograms for two older Intel Xeon CPUs. Using a modern AMD EPYC CPU, we would expect the MVM execution time to decrease. This fairly natural partitioning of the MVM uses only five CPU cores; using additional CPU cores would further decrease the execution time.

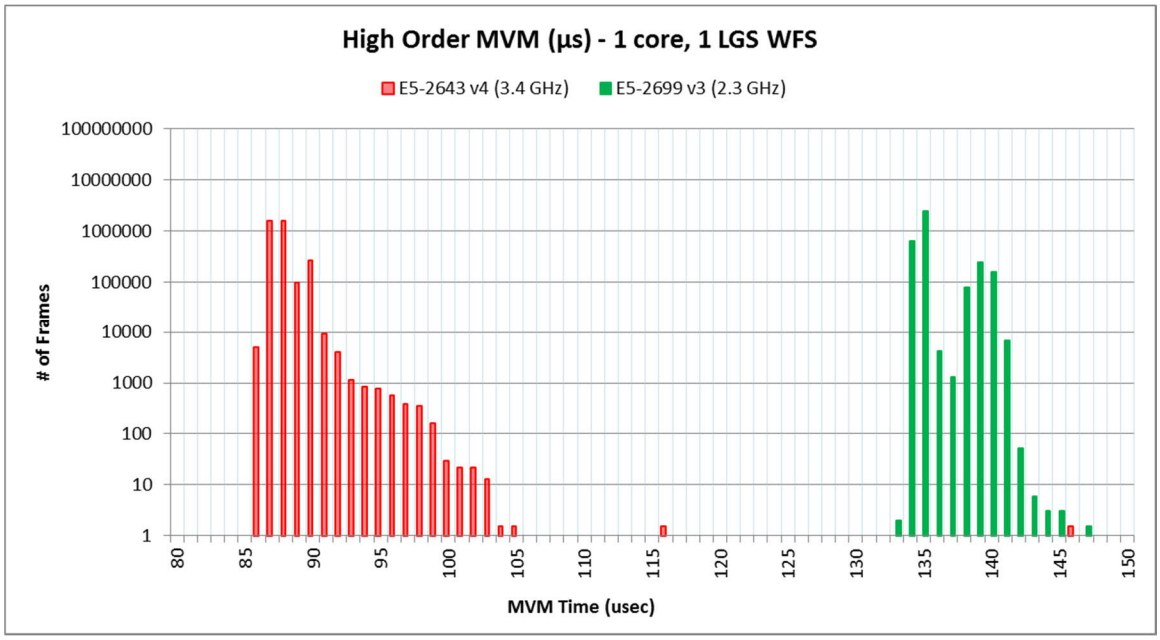


Figure 14-5- Execution Time Histograms

During NFIRAOS RTC benchmarking, there was significant concern about using CPUs for the computationally demanding RTC tasks. In order to verify that it is possible to meet the NFIRAOS RTC performance requirements, a simplified round trip benchmark was written in which pixels from two LGS WFS quadrants were sent from a test server to an RTC HOP (High Order Processing) server. Only two of the four quadrants were sent since the receiving server contained only two CPUs but, would require four CPUs to perform the required MVM computation. The pixel stream was sent at a rate of 800 frames per second, with the actual data being sent over a 500 µs interval to mimic the LGS WFS readout timing. Full frame (all four quadrants) were sent in a separate test to verify that the HOP server could read and process the full pixel stream.

The HOP server read and calibrated the pixels, computed gradients using match filters, performed the MVM corresponding to two LGS WFS quadrants and sent back computed DM error vectors corresponding to each quadrant (the HOP server will sum the vectors produced from each quadrant but, in the benchmark, each vector was sent back to the test machine to make the Ethernet load closer to the full system). Once every ten seconds, the HOP server would swap to a different control matrix to simulate the application of a newly optimized control matrix.

The NFIRAOS RTC must complete the sending of the DM vectors to the DM electronics within 1200 µs of the arrival of the first LGS WFS pixel. Using a first generation E5-2670 based server, the RTC server was not able to operate at the required 800 Hz. Many frames would be processed within the required 1200 µs but, once a frame required extra processing time, it would trigger a cascade affecting every later frame.

Table 5-3 below, shows the CPUs used for the round trip benchmark. Each CPU used in the benchmark contains 2.5 MiB L3 cache per CPU core. The E5-2643 v4 based server was not used for the round trip benchmark; it was purchased as a prototype server for the post MVM processing and the real-time storage and does not have enough cores or L3 cache (20 MiB) to perform well in the role of the HOP server. For the sake of comparison, three more recent CPUs are included in the table; an Intel Gold 6240 (24.75 MB L3), AMD EPYC 7702 (256 MiB L3) and an AMD EPYC 7402 (128 MiB L3). AMD currently has a significant advantage due to the higher core count and much larger L3 cache. Significantly better performance is expected using more modern CPUs.

Table 14-3 - Round Trip Benchmark Results

Processor	Released	# Cores	Base clock (GHz)	Memory GB/s	Mean (µs)	St. Dev. (µs)
E5-2670	2012 Q1	8	2.6	51.2	1140	7.2
E5-2697 v2	2013 Q3	12	2.7	59.7	890	16.5
E5-2697 v3	2014 Q3	18	2.3	68	597	6.0
E5-2643 v4	2016 Q1	6	3.4	76.8	N/A	N/A
Gold 6240	2019 Q2	18	2.6	140	N/A	N/A
EPYC 7702	2019 Q3	64	2.0	204.8	N/A	N/A
EPYC 7402	2019 Q3	24	2.8	204.8	N/A	N/A

Figure 14-6- Round Trip Benchmark Histograms below shows timing histograms from these round trip benchmarks. In the case of the E5-2699 v3 server, the entire control matrix fits within the L3 cache and the timings have little timing jitter. The test server usually receives the DM vectors within ~600 µs of the sending of the first pixel packet for the frame. On the E5-2699 v3 server, when a new control matrix is used, the MVM takes an additional ~240 µs due to fetching the control matrix from RAM. This difference would be significantly reduced with a modern CPU with greater memory bandwidth.

Including the round trip timings is meant to show how we have benchmarked a significant portion of the critical path but the standard deviation timing is quite small (e.g. 6 µs) compared to the 100 µs requirement.

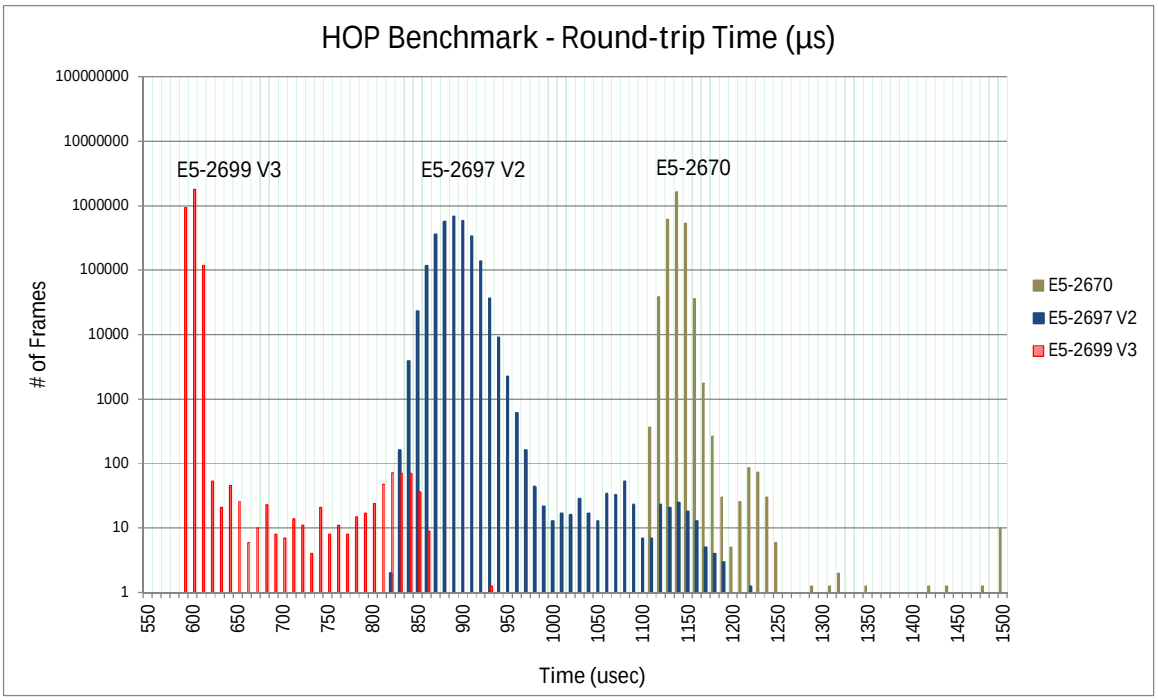


Figure 14-6- Round Trip Benchmark Histograms

Figure 14-7 below shows an estimated timing diagram for the NFIRAOS RTC using Xeon Gold processors, which were the baseline design prior to the release of the AMD EPYC 7002. The round trip benchmark above roughly corresponds to all of the steps up to and including transporting the DM vectors. The MVM time was estimated by factoring in the memory bandwidth for the Xeon Gold versus the older Xeon E5 CPU. The subsequent steps were either benchmarked directly on the E5-2643 v4 server (e.g. DM clipping) or estimated based on benchmarks of similar operations (e.g. sparse MVM for DM extrapolation).

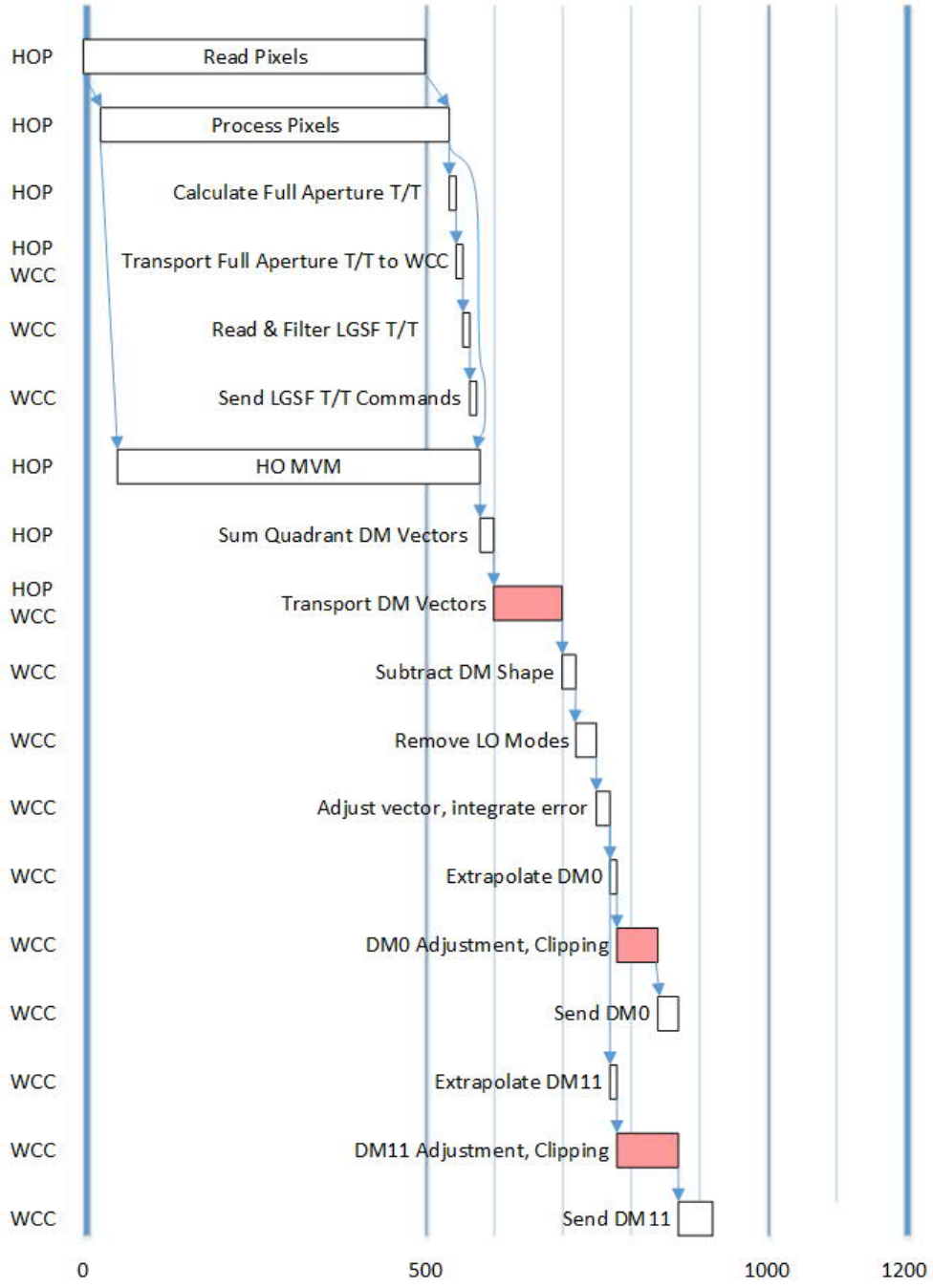


Figure 14-7 - Timing Diagram for NFIRAOS RTC

Using the measured benchmark results and estimated performance for various pipeline stages, a Monte Carlo simulation was produced to show the expected timings for the six HOP servers and the WCC (wavefront corrector controller) server which accepts the intermediate DM error vectors and computes the final wavefront corrector vectors. As shown in Figure 14-8 below, even in the case of the NFIRAOS RTC, if the control matrix is cached, the wavefront corrector commands are sent to the devices within 300 µs of the last LGS WFS pixel arriving. With the large L3 cache provided by the EPYC 7002 CPUs, the control matrix can be pre-fetched and the commands would be sent within 300 µs of the last WFS pixel arriving even when a control matrix was swapped.

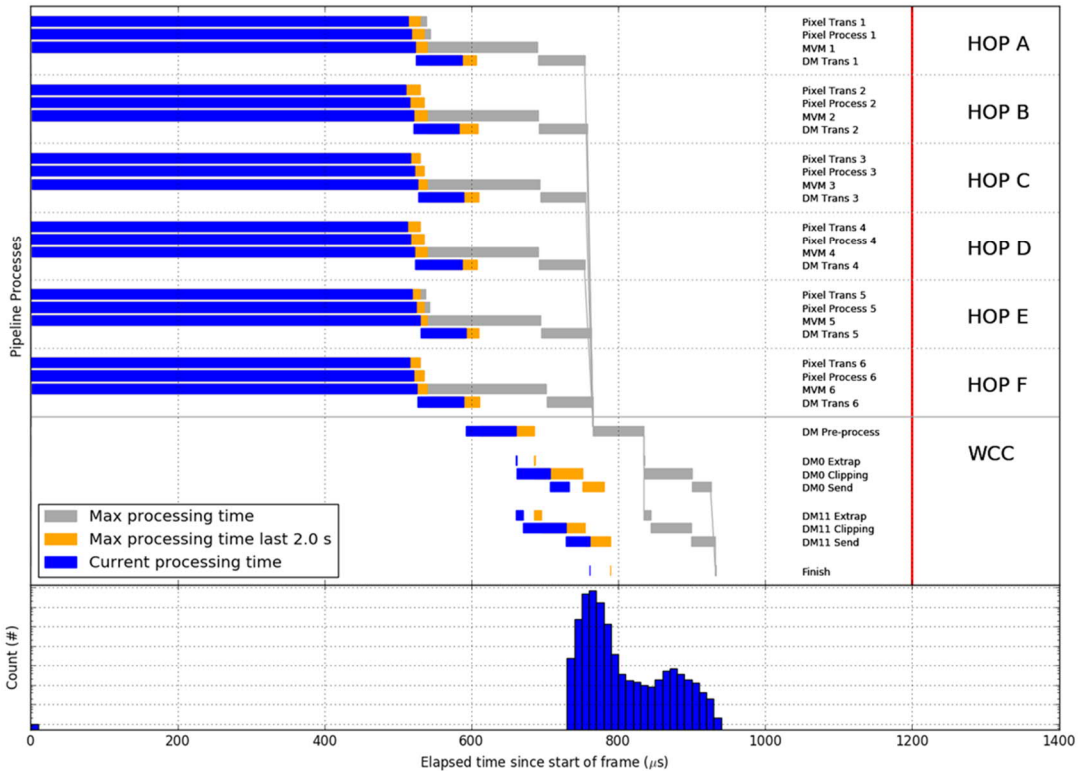


Figure 14-8 - Monte Carlo Timing Simulation

14.2 PERFORMANCE COMPLIANCE

Based on the benchmarks of Section 14.1.1 we are confident that the GNAO Template RTC will comfortably achieved the 1 kHz loops rate (requirements 8.3.1.1 and 8.3.2.1, see Table 8-2). With the reduction of five LGS WFS to four and three DMs to one, this gives us an even greater safety factor and make achieving the 2 kHz loop rate goal even more likely.

Achieving the 500 µs latency from the arrival of the last pixel to the application of the DM and TTS commands (requirements 8.3.1.2 and 8.3.2.2, see Table 8-2) is mostly shown by the round trip benchmark in Table 14-4. Similarly, the 100 µs timing jitter limit (requirements 8.3.1.3 and 8.3.2.3, see Table 8-2) is also demonstrated in same figure, though there are some caveats that are covered in the next section.

14.2.1 Detailed Performance

The computational requirements of the RTC NGS processing loop are significantly lower than those of the RTC LGS processing loop. As shown in Section 14.1.1, even using hardware that is now several years old, the main computations and network latency that drive the critical path for the RTC LGS easily meet requirements. However, the benchmark shown there does not include some additional calculations, so we provide additional explanation and estimates of their impacts here.

Table 14-4 – Detailed RTC timing estimates

Pipeline Stage (operations)	Included in **Round-Trip** benchmark?	Notes	Time Estimate (µsec)	Cumulative Latency (µsec)
Read pixels	Y	1,2	have	500
Process pixels (calibrate, compute gradients & flux)	Y	1,2	3.7	503.7
High Order Reconstructor				
- DM shape to LGS gradients sparse MVM	N	3	50	N/A
- High Order MVM	Y	4	34.8	538.5
High Order Partial Vector Combination				
- Sum quadrant DM Vectors	Y	5	1—2 us	540.5
Temporal Filtering & Combination	N	6	10	550.5
Closed Loop Wavefront Correction	N			
- DM adjustment & clipping		7	30	580.5
- Send DM actuator commands	Y	8	15	595.5
- Contingency	Y	9	50	645.5

1. It is assumed that the high order LGS WFS requires 250-500 µs per frame to digitize and transmit LGS WFS pixels to the RTC. We take 500 µsec as the reference for the remaining latencies in this table.
2. Based on measured pixel processing times in the Round-Trip benchmark.
3. NFIRAOS sparse MVM benchmark (we have HEART code but not as a benchmark)
   NFIRAOS LGS mode, 5792 gradients, 41 non-zeros per row, 168 µsec.
   GNAO has only ~608 gradients per WFS, but uses three DMs rather than two so there will be more non-zeros per row. NFIRAOS benchmarks assumed 16 + 25 non-zeros per row. If we assume at most 36 non-zeros per DM for each gradient, then we would have 50. Note however that this is not on the critical path, and does not add to the latency, since this operation can be performed after the DM commands are sent and before the next frame arrives.
4. Based on measured MVM times using single threads in the Round-Trip benchmark.
5. Estimate scaled down from NFIRAOS benchmarking.
6. Assumes that the LO data have already arrived. Computation time estimate scaled down from NFIRAOS benchmarking.
7. Pessimistic estimate, extrapolating from NFIRAOS results that used larger DMs.
8. Previous testing has found one-way network latencies of 10 µs on a well-tuned system. For NFIRAOS-sized DM commands it took 30 µs. Scaling down to GNAO, we estimate ~5 µs, but add 10 µs for the minimum latency, for a total of 15 µs.
9. The pessimistic value taken from the Round-Trip test includes about 50 µs of extra latency that was probably due to the start time for the RTC measurements being based on the time the first

pixels were sent from the WFS pixel simulator, rather than their arrival at the RTC. The network interfaces were also not fully tuned. We add this time as contingency.

Table 14-4 combines measurements of the benchmark testing in Section 14.1.1 with estimates for quantities that were not measured in the round-trip timings based on NFIRAOS RTC prototyping. We feel that the timings shown here are pessimistic, as we have erred on the side of caution for quantities that we did not measure, and we are using test hardware that is not as powerful as what we plan to use for the real system. Despite this, we easily meet REQ-8.3.2.3 that states that the latency between the last LGS WFS pixel arriving and last DM actuator being sent is less than 500 µs. Subtracting 500 from the last row gives our estimate of 146 µs.

In order to meet requirement REQ-8.3.2.3, the RTC LGS processing loop latency must have a standard deviation less than 100 µs. The standard deviation for the Round-Trip test was shown to be 12.2 µs, which is likely pessimistic due to the fact that it includes jitter in the pixel send times. Exceeding the requirement in this benchmark by such a large factor ensures a large margin for additional tasks that were not implemented in the benchmark.

In conclusion, we are very confident that the Gemini Template RTC will easily meet both the NGS and LGS requirements related to control loop timing jitter and latency.