A Real-time Control Computer for the E-ELT

Document: GF-PDR-04

Top Level System Architecture

Version 1.5

18th January 2016

Change Record

Version	Date	Author(s)	Remarks
0.1	1 Oct 2015	Nigel Dipper	Initial skeleton version
0.3	9 Oct 2015	Nigel Dipper	Input from Eddy Younger
0.4	8 Dec 2015	Urban Bitenc	HW&SW options to investigate
1.0	7 Jan 2016	Nigel Dipper	Restructured for PDR – First draft
1.0.1	8 Jan 2016	Nigel Dipper	Microgate Logo Corrected
1.1	8 Jan 2015	Nigel Dipper	Order of sections changed
1.2	12 Jan 2016	Urban Bitenc	2.1: restructured, added new diagrams and text. Fixed numbering of figures and sections. Update the table of abbreviations.
1.3	13 Jan 2016	Nigel Dipper	Updated Figure 1 and Figure 2. Some text changes
1.4	16 Jan 2016	Nigel Dipper	FPGA text from Deli added. All sections complete. Still requires checking before release for PDR
1.5	18 Jan 2016	Nigel Dipper Urban Bitenc	Configuration diagrams updated. AD document list corrected. Version for release for PDR.

Table of Contents

Acronyms and abbreviations

AD	Applicable Document
AO	Adaptive Optics
CANARY	Durham/LESIA on-sky AO demonstrator
CPU	Central Processing Unit
CUDA	NVIDIA GPU based software development language
DARC	Durham AO Real-time Controller
DDS	Data Distribution Service
DM	Deformable Mirror
DRAGON	Durham laboratory-based AO demonstrator bench
ELT	Extremely Large Telescope
E-ELT	European ELT
ESO	European Souther Observatory
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HLS	High Level Synthesis
HPC	High Performance Computing
MIC	Many Integrated Core
MVM	Matrix-Vector Multiplication
NIC	Network Interface Controller
PCIe	Peripheral Component Interconnect express
RD	Reference Document
RTC	Real-Time Control
RTL	Register Transfer Level
SIMD	Single Instruction Multiple Data
SPARTA	ESO VLT AO Real-time Control System
SHERE	VLT Planet finder instrument
UDP	User Datagram Protocol
UK ATC	United Kingdom Astronomical Technology Centre
VLT	Very Large Telescope
WFS	Wave-Front Sensor
WP	Work Package

Applicable Documents

These are the Green Flash PDR documents.

No.	Title	Reference	Issue	Date
AD01	Introduction to Green Flash	GF-PDR-01		
AD02	Management plan and WP definition	GF-PDR-02		
AD03	Requirements Specification	GF-PDR-03		
AD04	Top Level System Architecture	GF-PDR-04		
AD05	Distributed GPUs for Real-time HPC	GF-PDR-05		
AD06	FPGA Solution for hard Real-time	GF-PDR-06		
AD07	Smart Interconnects	GF-PDR-07		
AD08	Interface Definition Document	GF-PDR-08		
AD09	Supervisor design	GF-PDR-09		

Reference Documents

These are documents external to the Green Flash project

No.	Title	Reference	Issue	Date

1 Scope

This document specifies the top level architecture for both the software and hardware of GreenFlash.

As the project objective is to find an optimum design for a prototype system, this architecture will evolve significantly during the course of the project. This document therefore only specifies the high level systems, sub-systems and some modules of the design and indicates how they may be interconnected. It defines the hardware and software options that will be investigated leading to the down-selection for the final prototype system.

2 Hardware Architecture

The starting point for the overall hardware architecture is the existing distributed design used for SPARTA. The overall design consists of 3 systems:

- Hard real-time data pipeline
- Soft real-time system
- Simulator system

The hard real-time data pipeline must meet the requirements for high throughput, low latency and low jitter (high determinacy). It takes as input pixel data from multiple cameras and provides as output the drive data for multiple deformable mirrors.

The soft real-time system provides an HPC facility for the configuration, calibration, control and optimisation of the hard real-time pipeline.

The simulator system provides accurate simulation at real-time frame rates of hardware components in their absence, such as WFS cameras and deformable mirrors.

The top level system engineering diagram for the full system is shown in a very generalised form in Figure 1.

Top Level System Architecture

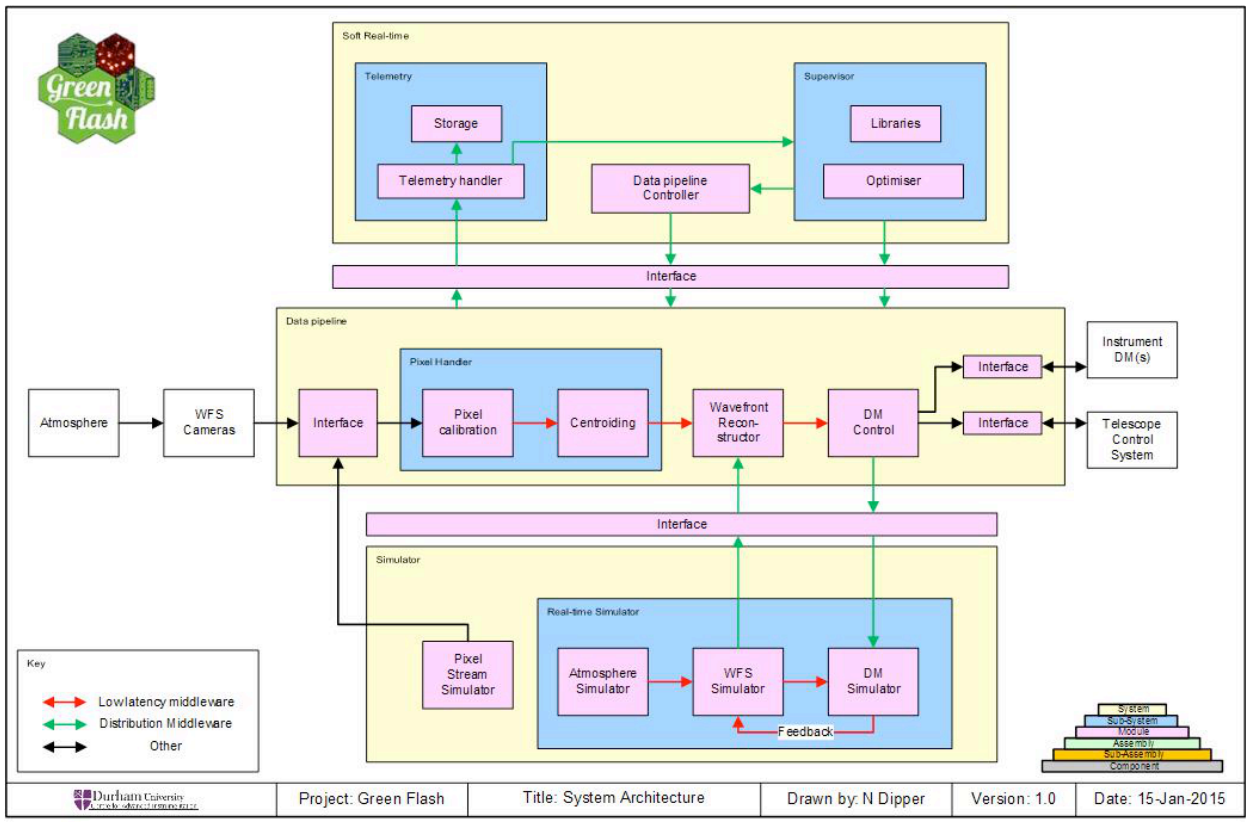

Figure 1: Top Level System Engineering Diagram.

2.1 Real-time Data Pipeline System

This is the low latency, low jitter data pipeline. The system consists of four separate sub-systems that will likely run on the same computer hardware (e.g. to provide low latency shared memory middleware) or could be distributed to multiple hardware systems via data switches. The sub-systems are:

- Pixel Calibration
- Centroiding
- Wave-front reconstruction
- DM controller

The down-selection of hardware for the real-time data pipeline is a part of the project and several different hardware options will be tested. These include: Field Programmable Gate Arrays (FPGA), Graphics Processing Units (GPU), Many Integrated Core processors (MIC) and conventional CPUs. There are two possible configurations for the data pipeline:

1. A CPU host that is a part of the pipeline with hardware accelerators likely accessed over PCIe
2. A fully embedded system with data input and output directly to/from the pipeline hardware with the host CPU system not a part of the pipeline

These two configurations are shown in cartoon form in Figure 2 and both configurations will be investigated. In both cases, there are various options for the embedded or acceleration hardware and for its configuration.

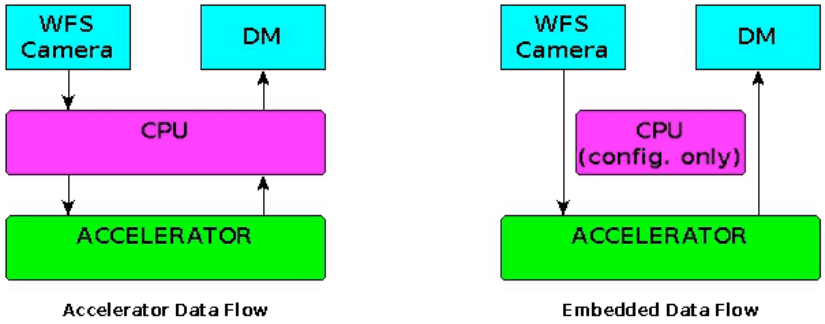

Figure 2: Alternative data pipeline hardware configurations. The arrows represent the flow of data.

The hardware that will be tested must be highly parallel and provide sufficient numbers of concurrent processing units/coprocessors operating in parallel to meet the system requirements. The hardware and configuration options that will be tested are outlined below:

2.1.1 Accelerator Hardware Options

The pixel calibration, centroiding and wave-front reconstruction by matrix-vector-multiplication (MVM) are all highly parallelizable processes. For a large system, they can be performed much faster (compared to the CPU) on hardware that is designed specifically for parallelizable algorithms. We refer to this hardware as “accelerators”. In this architecture, computationally intensive tasks are offloaded to accelerators.

The accelerators to be investigates are GPUs, MICs (Xeon Phi) and FPGAs. Whilst all of these technologies can provide parallel processing, the architectures are significantly different with advantages and disadvantages for this application. The FPGAs bring an additional advantage in providing determinacy, reducing the resultant jitter to close to zero.

Separate prototype systems will be constructed to demonstrate the relative capabilities of each accelerator. The functionality of each system will be the same.

In the following we briefly describe the specifics of each accelerator. In sections 2.1.2 - 2.1.4

we then discuss different architectural options for the use of these accelerators.

2.1.1.1 GPU accelerators

GPUs provide massively parallel systems (100s to 1000s of processing units) that are specifically designed for applying single instructions to multiple data in parallel (SIMD). As such they are very powerful technology for performing the sort of linear algebra that is required in an AO RTC. This technology will be investigated as a candidate in all three of the main Green Flash systems: Data Pipeline, Soft Real-time and Simulator. The investigation of distributed GPUs for real-time HPC is a major work-package in Green Flash and is described in detail in RD05.

2.1.1.2 Xeon Phi accelerators

Recent years have seen the emergence of Many Integrated Core processors as a competing hardware acceleration technology to GPUs. The most common example is the Xeon Phi. These devices sacrifice some of the complex functionality of the multiple cores found in standard CPUs in order to provide up to 60 (in current devices) cores on a single chip. The devices are currently packaged in a similar format to GPUs, on an extension card accessed over PCIe with correspondingly high data transfer rates. Unlike a GPU however, they can act as stand-alone devices with all of the application code running on them rather than partially on a host CPU system. The road-map for these devices may see them used as the standard processor on computer server mother-boards.

The advantages of the Phi over GPUs are:

1. They are programmed in C with some extensions to support their large scale parallel nature. Thus code written by application programmers in C can be readily ported to these devices. There is no specialist development language, such as Nvidia CUDA for GPUs, that must be learnt by the programmer. The development cycle is thus very fast.
2. Whilst the many cores can be applied in a SIMD configuration as is done with GPUs, there is the option for sub-sets of the cores to perform separate and independent tasks. This provides an additional flexibility that is absent from GPUs.

A Green Flash work-package will draw comparisons in the performance of these devices against that provided by GPUs (and indeed FPGAs). Just as in CPU devices, they are susceptible to system jitter albeit at a lower level.

Initial investigations of the performance of these devices has already started at CfAI in Durham (in collaboration with the UK ATC) with encouraging results. A full prototype RTC with MIC acceleration will be tested using existing devices and can be expanded to include the stand-alone version (Knight's Landing) that is expected to emerge during 2016.

2.1.1.3 FPGA accelerators

Whilst FPGAs accelerators are an option for the data pipeline, the advantages in determinacy (low system jitter) that they provide are more likely to be employed in an embedded configuration than as an accelerator. This embedded FPGA data pipeline option is described in AD06. FPGAs also represent an attractive option for interfacing both between the various systems and modules and to/from external WFSs and DMs. This 'smart interconnect' concept is described in AD07.

However, new FPGA development tools, and particularly the PLDA Quickplay tool, also make FPGA accelerators an attractive option for algorithm acceleration in the soft real-time optimiser sub-system and as a part of the real-time simulator system and these options will be investigated. A brief introduction to FPGAs and the tools now available for their programming is given here:

FPGA Background

The modern FPGA has developed far beyond a cluster of programmable gates. It normally integrates with high end digital components, such as multi-gigabit transceivers, high speed memory interface, the latest I/O interfaces and controllers (e.g. PCIe), embedded DSP processing units (hardware multiplier), embedded memory blocks etc. All these make FPGA a flexible yet powerful digital processing solution especially for those parallel tasks with hard real-time requirements. However, programming of FPGA can be a slow and costly procedure. This is partially due to the customized logic and the parallel complexity of the system where timing always plays a vital role. Moreover, the tools addressing FPGA development are traditionally focused at the RTL level (Verilog or VHDL), which has analogies to assembler and lacks the development efficiency of high level compilers.

Debugging FPGA logic is another lengthy and frustrating process. Simulation is often used by the engineers to verify the design logic. There are usually two kinds of such simulation: functional simulation and timing simulation. The functional simulation is to test that the designed FPGA logic works to the required specification. The timing simulation tests that the physical speed of the FPGA can be achieved. Luckily timing tools are provided by the FPGA vendors. In most designs, timing is statically analysed and a logic analyser can be used to test the FPGA logic in real-time.

All of this complexity is gradually being abstracted from the programmer by high level development tools. The evaluation and use of these tools is critical to the use of FPGA technology in Green Flash.

FPGA programming Methodology

Nowadays there several different approaches to FPGA development. These are identified here. As with any technology, each option has its own features and limitations. Green Flash will identify and verify the best options to use for higher level FPGA development to improve productivity. Although all of these methods may have a role to play, the project will concentrate mainly on demonstrating the efficacy of the PLDA Quickplay tool.

1. Register-Transfer Level (RTL)

The use of the traditional RTL based design method should not been overlooked. Whilst it does not provide the powerful development environment that we require, it can be very useful in interfacing standard IP cores very efficiently and can play a role in even where higher level methodologies dominate.

2. High Level Synthesis (HLS)

   The claim for HLS is that the FPGA can be programmed with a popular programming language, normally a limited version of C. Xilinx Vivado HLS is one such option. Whilst this is a powerful technique, it does not provide the full abstraction that allows the non-specialist to easily produce FPGA code. The PLDA tool Quickplay (see below) builds on top of HLS. It thus takes the final step to providing an easy development methodology. It is this highest level of abstraction that will be investigated by Green Flash.

3. OpenCL

   OpenCL is an initiative to provide cross-platform and even cross-architecture computing acceleration. Similar to the Nvidia GPU CUDA architecture, the application logic is built as blocks of kernel and is uploaded to the accelerator. The data is provided in a form of buffer and the result is also transferred from the accelerator to the host in a buffer. Although the portability of the acceleration is a very promising feature, the buffer based data transfer does not match the stream based architecture of, for example, the pixel stream handing of an AO real-time system.

4. PLDA Quickplay

   PLDA Quickplay has the potential to provide a fully abstracted development environment for FPGA software. Its architecture is especially well suited to the design of an AO RTC system. Firstly, it brings up a system architecture based on the Kahn Process Network (KPN), which connects multiple processing kernels via FIFO based stream interfaces. This model matches the data flow of an AO RTC. Secondly, Quickplay provides a HLS system allowing the application model to be described in C style programming language allowing much of the functionality to be simulated and debugged with readily available C tools. Furthermore, Quickplay is an integrated environment. The host system interface and the standard communication interface are all provided and tested. The developer can concentrate almost entirely on the user application. A diagram of the QuickPlay work flow (from www.quickplay.io) is shown in Figure 3.

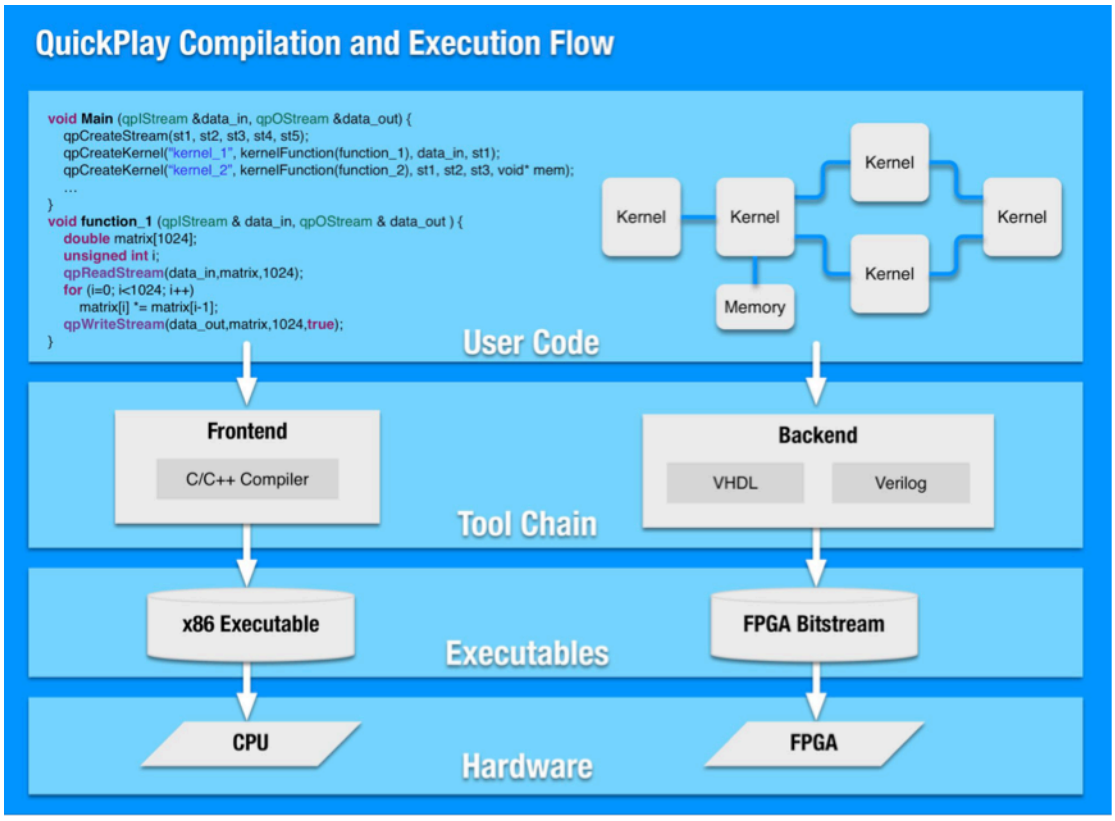

Figure 3: PLDA Quickplay FPGA development methodology

The core of the FPGA application can be built in the C programming language and go through the familiar test and verification procedure as seen in the left vertical path, after which the design can go through the HLS and system component integration (right vertical path) to automatically generate the FPGA bitstream with the same functionality but with the desired real-time performance and determinism.

Whilst traditional FPGA tools will be used where required, most of the FPGA based acceleration architectures will be targeted with Quickplay

2.1.2 Accelerator Configuration Options

Whichever of the acceleration technologies discussed above is used, there are various options for how much of the data pipeline is migrated to the accelerator and how much remains in the CPU. We intend to test the options described in the following sections in order to provide a clear understanding of the advantage of each configuration. The options in this section represent the baseline accelerated system with no smart interfaces. Pixel data are captured by the CPU. Then there are three options:

(a) no acceleration. The full data pipeline is implemented in CPU.

(b) pixel calibration and centroiding are performed on the CPU and centroids are copied to the accelerator (section 2.1.2.2, Figure 5), or

(c) pixel data are copied to the accelerator, where then the pixel calibration and centroiding are performed (section 2.1.2.3, Figure 6).

In (b) and (c), the MVM is performed on the accelerator. This is because, for systems at ELT scales, the wave-front reconstruction by MVM dominates the computational requirements. In both cased the DM commands are copied back to the CPU and sent to the DM.

While the accelerator performs the computations faster, the additional steps of copying the data to the accelerator and copying the result back to the CPU add to latency: the overall gain in time is the difference between the speed-up time and the data-copying time. Data copying may also increase jitter. The object of investigating the use of this accelerated architecture is to demonstrate the performance levels that can be achieved and the advantages and disadvantages of this option.

2.1.2.1 CPU-only option

This option, with no hardware acceleration, is included as a baseline for comparison purposes and is illustrated in Figure 4. The data pipeline will be implemented in C on a top end server with typically 2 off 6 core processors. The system will be multi-threaded (up to 12 threads or 24 if hyper-threaded). Whilst such systems are common in low order AO systems on 4m and 8m class telescopes, the throughput will naturally be limited for ELT scale systems and unable to meet the requirements. However, we will demonstrate what can be achieved with 'pure-CPU-systems'. The maximum frame rates that can be achieved will be measured along with the jitter. The latter will depend critically on the Linux kernel used. This system will allow us to demonstrate what can be achieved using non-real-time and real-time kernels. These results will demonstrate the sort of acceleration factors that must be achieved with the accelerated and embedded options.

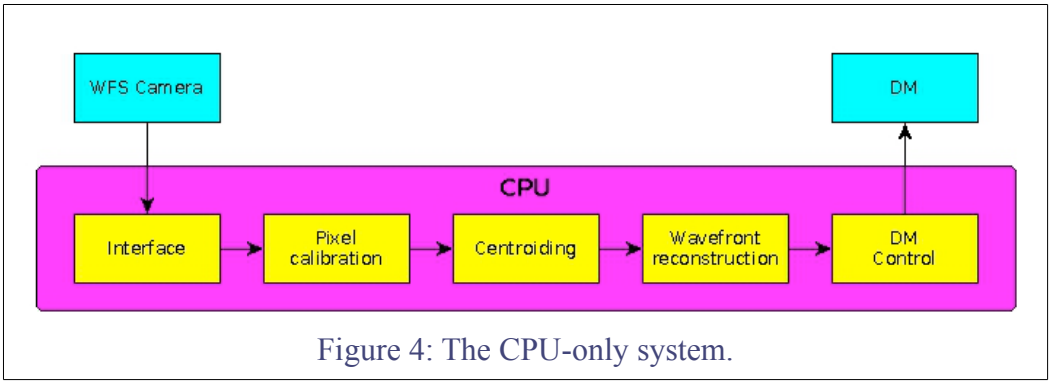

Figure 4: The CPU-only system.

2.1.2.2 Only MVM on the accelerator

For a large AO system, wave-front reconstruction by MVM takes about 70%-80% of processing time in each cycle. By offloading this task to an accelerator as illustrated in Figure 5, a significant speed-up can be achieved. The amount of data that needs to be copied from

the CPU to the accelerator on each cycle is small: the centroids copied consist of 2*N floating point numbers, where N is the number of sub-apertures. Hence the acceleration of MVM easily makes up for the time lost due to the extra data-copying.

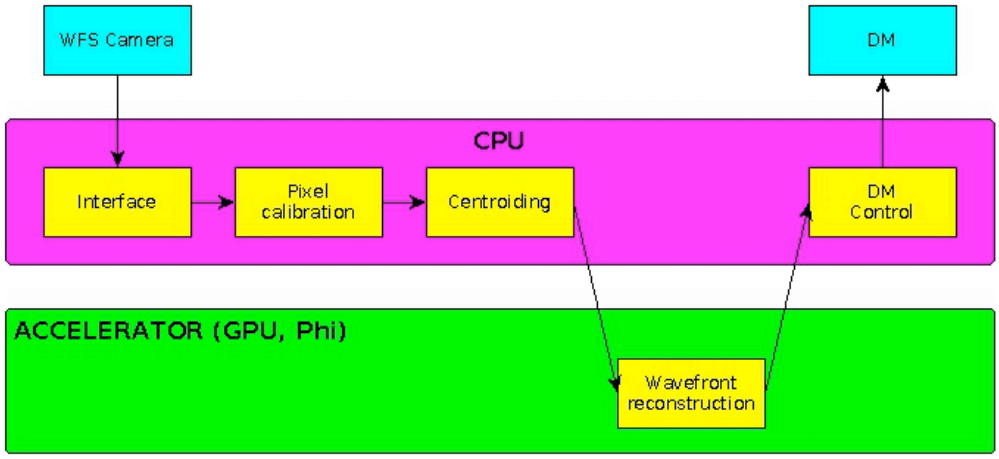

Figure 5: MVM, which is the most computationally demanding part of the data processing pipeline, is processed by the accelerator.

2.1.2.3 The whole pipeline on the accelerator

Performing all the processing steps on the accelerator as illustrated in Figure 6, has the advantage of speeding-up also the pixel processing and centroiding which are both highly parallelizable operations. However, the amount of data to be copied from the CPU to the accelerator is much larger with this architescture, at least *npix***N* floating point numbers. Here N is the number of subapertures and *npix* is the number of pixels in each subaperture, which for a laser guide star could be 16*16 or 20*20. To have the option of dynamic centroiding (where sub-aperture borders are re-defined on the frame-by-frame basis to follow the movement of the spot), one would need to copy entire frames to the accelerator and the amount of data would be even larger. This can lead to an increase of latency of a milisecond or more.

However, we note that the GPUs offer the possibility to perform a significant part of data copying in parallel to data processing, largely masking the copying time and reducing the increase in latency.

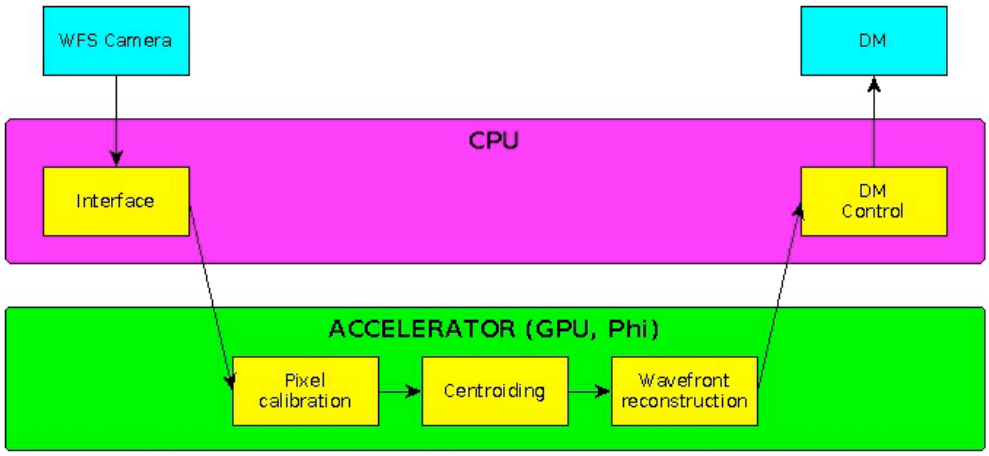

Figure 6: All the data processing happens on the accelerator and the CPU is used to interface the camera and the DM.

2.1.3 Accelerated System with Smart WFS Interface

This architecture is illustrated in Figure 7 and represents one step towards a fully embedded data pipeline by introducing a smart WFS interface. The pixel data are copied directly to the accelerator where they are processed. The DM commands are still copied to the CPU and sent to the DM. Note the accelerator could be a hybrid with, for example, pixel processing implemented in a FPGA and consequently MVM performed on a GPU.

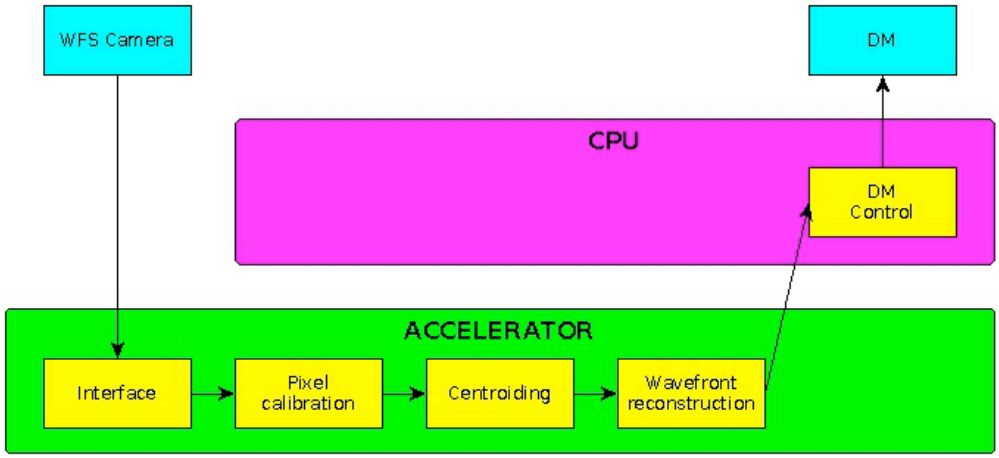

Figure 7: Pixel data are captured directly by the accelerator, by-passing the CPU.

Capturing the pixel data from the camera directly to the accelerator, by-passing the CPU has a big advantage of reducing both latency and jitter. The 'smart interface' can be provided by FPGA based technology in which some element of also pre-processing can be implemented. Leaving the DM interface CPU based and un-changed will allow us to investigate the advantage to be obtained in throughput and jitter from this intelligent WFS to accelerator interface alone.

2.1.4 Fully Embedded Data Pipeline

Adding a second interface directly from the accelerators to the DM results in what we refer to as a 'fully embedded' data pipeline where the CPU host is responsible only for configuration and control issues. This option represents the embedded data flow configuration of Figure 2.

As in 2.1.3, the pixel data flows directly into the accelerator from the WFSs, but now also the DM is controlled by the accelerator, avoiding the CPU completely. As illustrated in Figure 8, the CPU is used for configuration only further reducing both latency and jitter. However, controlling a DM directly from an accelerator would require development of specialised software and hardware, most likely in close collaboration with the manufacturer of the DM used.

The investigation of a fully embedded data pipeline, possibly including a hybrid of FPGA and GPU technology, is a major work package within Green Flash. The development of an FPGA based micro-server will be investigated by Microgate (see AD06) and the use of multiple distributed GPUs by LESIA (see AD05).

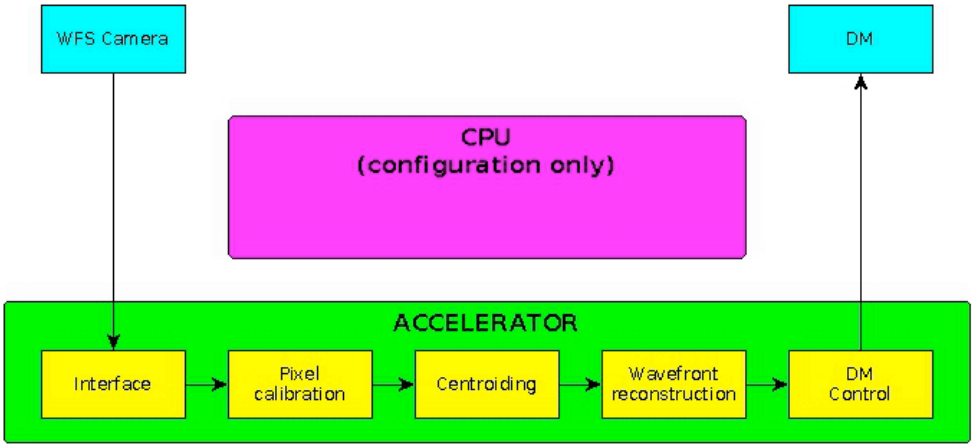

Figure 8: A fully embedded pipeline: the data by-passes the CPU completely.

2.2 Soft Real-time System

The soft real-time system is responsible for the configuration, calibration, control and optimisation of the hard real-time pipeline. Many of the algorithms required will require substantial computational power. This power is likely to be provided by technology similar to that used for the data pipeline. The configuration will likely be an accelerator data flow where a cpu host offloads large tasks to accelerator hardware.

The hardware requirements of the soft real-time system are defined by the functionality that can be seen in Figure 1:

1. The supervisor – This sub-system contains the primary functionality of the soft real-time system and is responsible for the configuration and optimisation of the AO data pipeline. The software makes use of data pipeline telemetry and external data (such as from turbulence profilers) to update the pipeline parameters including the control matrix. The principal requirement is therefore for an HPC system that is capable of performing the required calculations and updating the data pipeline on a timescale consistent with the changing atmospheric and telescope conditions. Since simulations have shown that such updates may be required by ELT scale systems as frequently as every 10 seconds, the HPC system will require acceleration hardware. The candidates for the hardware to provide the required acceleration are the same as those for the data pipeline (GPU, FPGA, MIC). The tools and techniques that are used

will thus be common to the development of both systems (and indeed to the simulator). The supervisor is a major work-package within Green Flash and is described in AD09.

2. Telemetry handling – All of the AO data generated by the data pipeline must be transferred to the telemetry system and made available to the supervisor and to the user interface as well as being stored. This is not a processing intensive application but requires hardware and middleware capable of dealing with very large data streams. It is highly likely that the data transport fabric will be Ethernet and suitable NIC devices will be investigated along with the associated switches. As is true throughout the RTC system, the concept of 'smart interfaces' will be applied to this data handling problem.
3. The data pipeline controller is responsible for sequencing and synchronising the pipeline. It handles all of the parameter updates whatever their source (user interface or supervisor). It has no requirements for processing or data handling outside of what can be provided by applications running on a standard CPU server.

2.3 Simulator System

The proposed simulator system is a development that has not been a feature of previous AO RTC systems. The top level requirement is to simulate elements of the AO system that are physically absent. This includes simulation of:

- Camera pixel data
- Wave-front sensors
- Deformable mirrrors
- The atmosphere (phase screens)

2.3.1 Simulator hardware options

Pixel data simulation allows cameras to be tested that are not present or may not yet exist! The data stream must be deterministic to correctly represent a camera pixel stream. This module is thus a prime candidate for FPGA technology. Pixel simulation will enable the exact latency and jitter of the RTC to be measure. An electronic signal can be fed back from various stages of the RTC to the simulator. This ability to measure system latency and jitter will be critical to determining the exact performance of both the data pipeline and the simulator in Green Flash.

The other modules will be simulated using the same or similar technology to that employed

for both the data pipeline and the soft real-time system. It will be possible to run the data pipeline with none, one or several aspects in simulation. Existing AO simulation code generally runs at much slower than real-time rates. Such code can be accelerated to close to the target rates of the data pipe-line using accelerator technology.

Since there is no president yet for such a system, Green Flash will investigate whether the simulator system should be in the form of a separate data pipeline linked to the primary pipeline or whether some form of CPU hosted and accelerated design can be employed.

2.3.2 Simulator to data pipeline interface

The pixel simulator will provide data to the pipeline via the same interface as real WFS camera. However, the form of the interface for data from the simulated WFS into the data pipeline and similarly for reconstructed data from the pipeline to a simulated DM will need to be defined. This problem is the subject of a work-package within Green Flash. System interfaces are described briefly in section 2.4 and are listed in AD08.

2.4 Data Pipeline Interfaces

The data pipeline has four interfaces and various options will be investigated for each. A detailed list and description of these interfaces is provided in AD08. A brief summary is given here:

2.4.1 Data Pipeline to WFS Interface Options

This interface carries the WFS camera pixel data from the camera (or camera simulator) to the pixel handling module. Whilst the pixel handling module is a part of the real-time data pipeline, all or part of the pixel handling may be implemented as a part of a 'smart interface' using FPGA technology. This interface may be located close to or remote from the actual cameras.

It is highly likely that the data transport for WFS camera data will be Ethernet and this will be the standard adopted by Green Flash. Legacy camera protocols such as Camera Link can be easily converted to Ethernet by commercial hardware and software so that existing non-Ethernet cameras can be used for testing when required.

Many commercial cameras use frame grabber hardware for this interface. This is not suitable for an AO RTC since it adds to the overall latency of the system. Camera data will be processed as the stream of pixels arrive. Thus the pixel processing is performed in parallel with the camera readout. This technique was used both in SPARTA and in the CANARY on-sky AO demonstrator.

For the option of a fully embedded data pipeline, a direct interface to the acceleration hardware is required. This may take the form of an FPGA based 'smart interface'. If the pipeline is heterogeneous and involves GPUs, a direct data path must be developed for the data stream using DMA over PCIe. This hardware interface technology will be investigated in Green Flash.

2.4.2 Data Pipeline to DM Interface Options

This interface carries the DM command data from the data pipeline system to the DM drive electronics. The data path is different depending on whether the target DM is within the telescope or is part of one of the instruments:

2.4.2.1 Interface to the telescope and hence to M4

The E-ELT telescope interface module is not yet well defined. The output of the real-time data pipeline will be in the form of DM commands and data that conforms to the definition of the telescope interface module. This will be a two-directional interface as feedback from the telescope systems to the real-time data pipeline will be required.

2.4.2.2 Interface to instrument DMs

Interfaces to different DMs (Alpao, Xinetics, Boston Micromachines, etc) are very specific to manufacturer and often only CPU, based drivers are available. If an interface is required directly from the acceleration hardware to the DM, this will require the development of drivers for that specific technology. One approach that will be investigated is the use of another FPGA based 'smart interface' that is addressed from the accelerator via PCIe and provides a DM specific interface.

2.4.3 Data Pipeline to soft-realtime system interface options

This is a two-directional interface. Telemetry data is published from the real-time data pipeline to the soft real-time system. Command data is sent to the real-time data pipeline from the soft real-time system for configuration, calibration, control and optimisation of the pipeline.

2.4.4 Data pipeline to simulator interface options.

This interface provides a facility to inject simulated data into the real-time data pipeline at various stages and to receive data from the actual real-time data pipeline into a simulated DM module. Aspects of this interface may be designed to be identical to the actual data pipeline interfaces to provide transparency between real and simulated data.

3 Software Architecture

Green Flash will provide a framework that facilitates the implementation of different control strategies within the data pipeline, different optimisation procedures within the soft real-time system and different techniques within the simulator. No detail is provided here of any of these software modules. At this stage, we simply identify the various software sub-systems and discuss the principles that will underpin the software development.

The factors driving the design approach include:

1. *Scalability* – the design will be scalable from small to very large systems.

2. *Abstraction* - a layered approach is required which abstracts the design logic from the implementation details such as the middleware used, the specific hardware and the programming language, etc. This abstraction allows all of the software to be highly portable.

3.1 Software systems/sub-systems

The software will consist of a number of modules that combine to provide the relevant system or sub-system. These are summarised in the following sections.

3.1.1 Data pipeline software.

The data pipeline software is the core of the AO system. However, it actually represents only a small fraction of the overall software required. Considerable experience in the design of real-time AO code exists at both LESIA and Durham. The software used for Green Flash, to compare and contrast the different hardware options, will draw on this experience and on this code base. One critical aspect will be the choice of low latency middleware for transporting the data through the various stages of the data pipeline. The existing Durham AO Real-time Controller (DARC) makes use of shared memory whereas SPARTA uses a low latency switch fabric. For an accelerated architecture, very similar code can be ported to the different accelerator hardware. For an FPGA based deterministic system, much of the code will be generated using Quickplay combined with other high level development tools.

3.1.2 Real-time simulator software

Several codes already exist within the Green Flash collaboration for end-to-end simulation of an AO system at ELT scales. The challenge is to modify one of these simulations to increase its speed to close to the frame rates of the actual data pipeline. The second is to design and implement an interface for the transfer of simulated data to the data pipeline and visa versa. The first stage of this work will be the definition of the interface. This will be done in the first instance with relatively slow existing simulation code in order to demonstrate the concept. Some work on on interfacing simulation code to a real-time data pipeline has already been done at both LESIA and Durham. Green Flash will build on this work to produce a full interface definition.

The process of speeding up simulations using hardware acceleration has also started. This work-package will investigate the same acceleration hardware discussed for the data-pipeline in section 2.1.1. As for the data pipeline, a hybrid of technologies may be used. Thus abstraction of the software will be a critical factor. Existing software has been written either in C or using CUDA when GPU hardware is targetted.

The final simulation software will be based on an accelerated version of the code from the COMPASS collaboration modified for compatibility with the Green Flash data interface.

3.1.3 Soft real-time software.

The soft real-time system is essentially an HPC system designed for the fast

processing and analysis of a large amount of data. This code is likely to be developed mainly in C with specialist accelerator modules where required. The development of the supervisor software that forms the heart of the soft real-time system is the subject of a major Green Flash work-package and is described in detail in AD09.

3.2 Ecosystem

The ecosystem is made up of software development tools, commercial and open source software libraries and code provided from external projects such as the ESO RTC system SPARTA, the Durham DARC system and the COMPASS software set. In addition, the transport of data throughout the system requires the evaluation, selection and integration of middleware.

3.2.1 FPGA Development environment

One of the major stumbling blocks for the use of FPGA technology is the slow development cycle for producing the 'IP cores' as FPGA firmware is known. However the tools for FPGA development have undergone very rapid evolution recently. Green Flash will wherever appropriate make use of the Quickplay tool being developed by PLDA. A brief introduction to this tool and to our approach to programming FPGAs is given in section 2.1.1.3.

3.2.2 Middleware

Communication between modules, sub-systems and systems will be achieved by carrying data over some fast data fabric such as 10G Ethernet using middleware. More than one middleware may be used in the system depending on the context. For example the telemetry system may use the data distribution service (DDS) or a similar commercial middleware application. However, within the data pipeline, minimum complexity and overhead is required. Since this is essentially point-to-point transport between functional modules of the pipeline, complex middleware is not required and so a simple high bandwidth low-latency option is preferred, such as shared memory, OpenMPI or simple UDP sockets.

As with all of the system software, the middleware will be abstracted both from the hardware and from the software that is the source and destination of the relevant data. In this way, any middleware selected can be easily superseded by an alternative when required.

The middleware testing and down selection is the responsibility of Durham and a part of the ecosystem work-package.

3.2.3 Algorithms and Libraries

Wherever possible Green Flash will make use of standard software libraries. This will allow us to take full advantage of the ongoing developments of such software, in particular the efficiency of these commercial packages in the utilisation of parallel processing architectures. Such mathematical libraries are now available that support CPU, GPU, MIC and even FPGA hardware. Whilst it is possible that alternative software developed in-house might provide higher efficiency for the specific application, shifting the burden of development (and thus risk) onto a product being developed elsewhere is a huge advantage. This principle will extend across all aspects of the Green Flash system. There is a sub-work-package for the

selection of appropriate libraries as a standard within Green Flash.

3.2.4 Re-use of existing RTC code - SPARTA

There is a substantial base of RTC code within the European AO community that can provide a starting point for much of the Green Flash software. As an example, the DARC system that operates both the CANARY AO on-sky demonstrator and the Durham DRAGON AO laboratory test bench has demonstrated powerful techniques for low latency data pipelining and the use of shared memory. Many of the techniques used within DARC are directly relevant for Green Flash and the full code is available to the collaboration.

One particular set of AO software that can make a substantial contribution to Green Flash is the soft real-time software developed by ESO for SPARTA. This code is written in C and meets all of the supervisor requirements for the existing VLT SPHERE instrument. This software can provide the ideal starting point for the Green Flash supervisor either to indicate a working design or, with the cooperation of ESO, for the actual re-use of much of the code. Since the RTC prototype that will be developed by Green Flash is targeted at the E-ELT, compatibility with the existing ESO VLT systems such as SPARTA at some level will be very beneficial. This concept is a part of the ecosystem work-package and will be investigated with ESO.

3.3 Software design principles

All software modules that make up the various systems and sub-systems should conform to standard design principles. So far as is practical, each module should contain:

- Core Logic - the core functionality of the module. This may itself be decomposed into a number of functional components
- A publish / subscribe interface (Telemetry)
- A command and control interface

3.4 Software Interfaces

Flexibility in the design concept is achieved through the use of well-defined and abstracted interfaces between components of a software module. External interfaces to modules are middleware dependent, and are implemented by middleware-specific components. The internal interfaces within a module are independent of the middleware.

Apart from the data pipeline itself, all interfaces throughout the system will implement a middleware-independent publish/subscribe interface via which a module may publish or subscribe to named data. A publisher or subscriber component is required for each named data entity (topic in DDS) which the module publishes or reads. The interface between the core logic and the publishers/subscribers must be independent of the middleware.

The details of the interface to the telescope control system are note yet defined. This illustrates the need for minimal coupling between such interfaces and the RTC core software in order for design and development of this software to proceed without the risk of extensive

rework when the interface becomes fully defined.

4 System integration and test

At the end of year 2 of the project, all of the separate options and prototypes described in this document should have been individually tested at the different institutions. At this point the down-selections will be made based on the results obtained and a decision will be made on the design for the final prototype system. The integration and test plan will be defined based on these down-selections and a full architectural design will be produced.

At this point we will enter a system integration and test phase. All of the relevant hardware and software will be integrated in Durham, making use of the laboratories there and some of the AO test bench elements such as wavefront sensor cameras, guide star emulators and deformable mirrors.

The integration and test work package (WP8) is the responsibility of Durham. System verification however will be the responsibility of Paris who will provide a verification plan and the final performance report