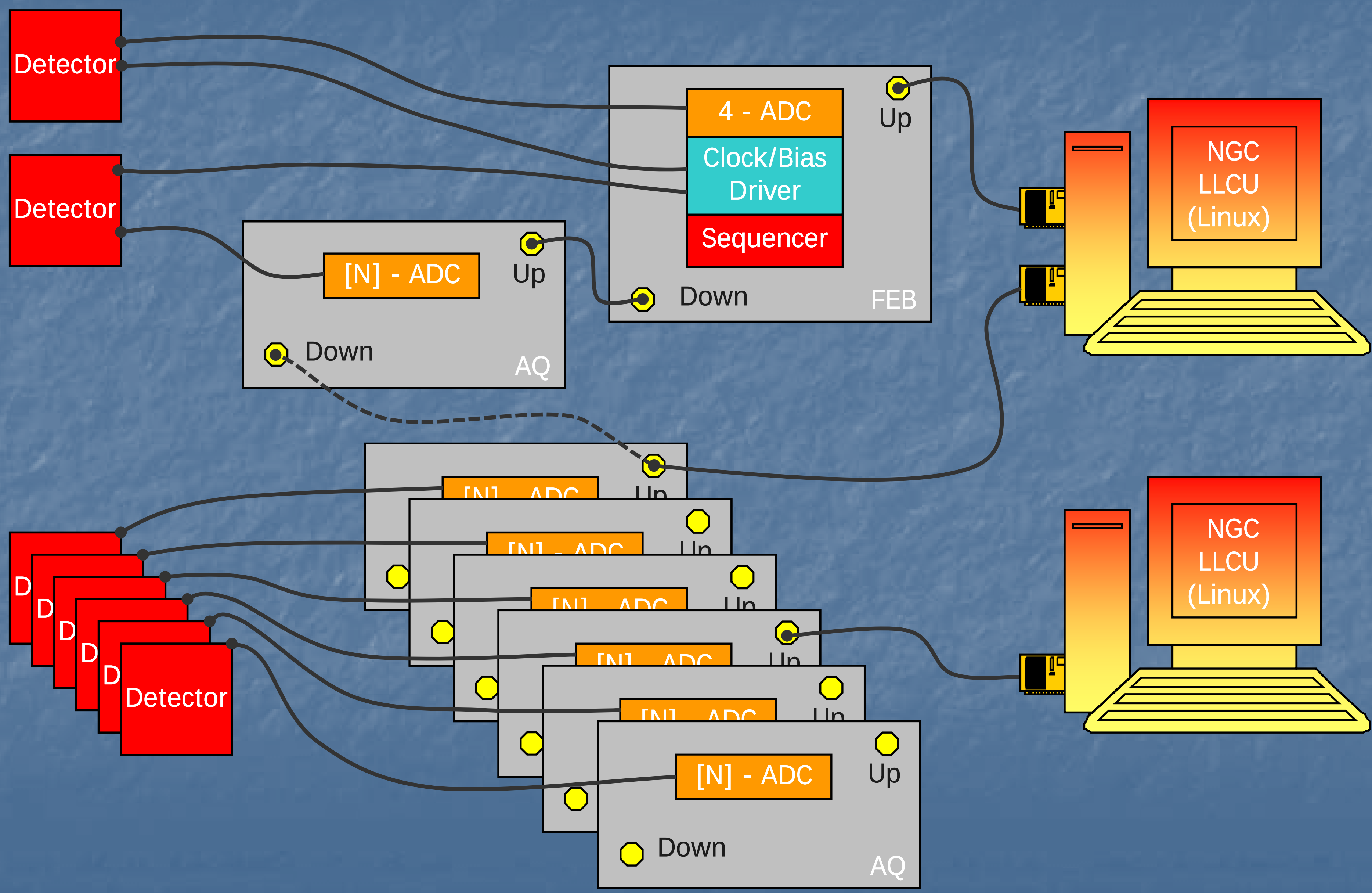
# ESO New General detector Controller (NGC)

## Base Software

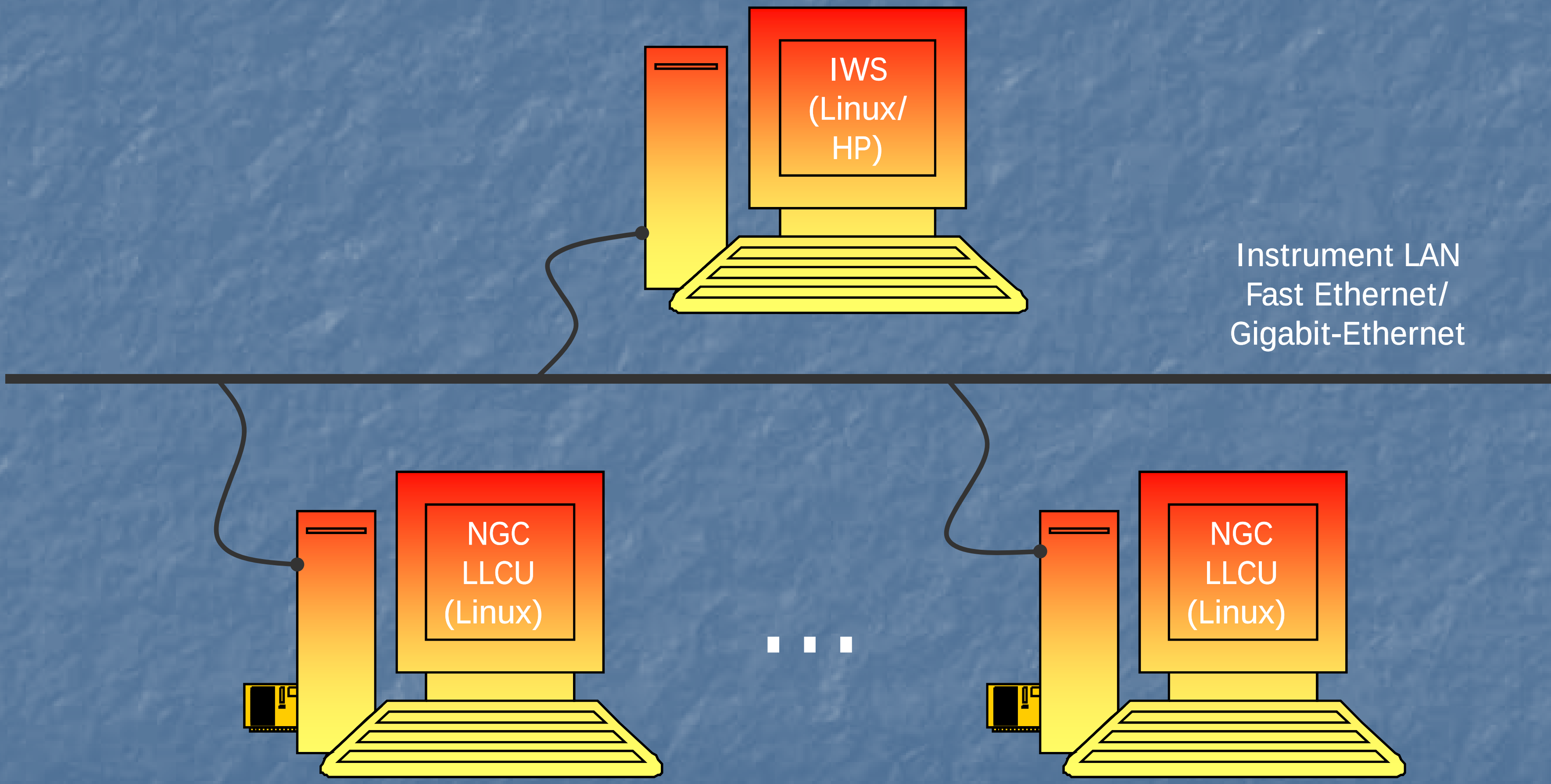
## And

## Infrared Detector Control Software

# System Overview

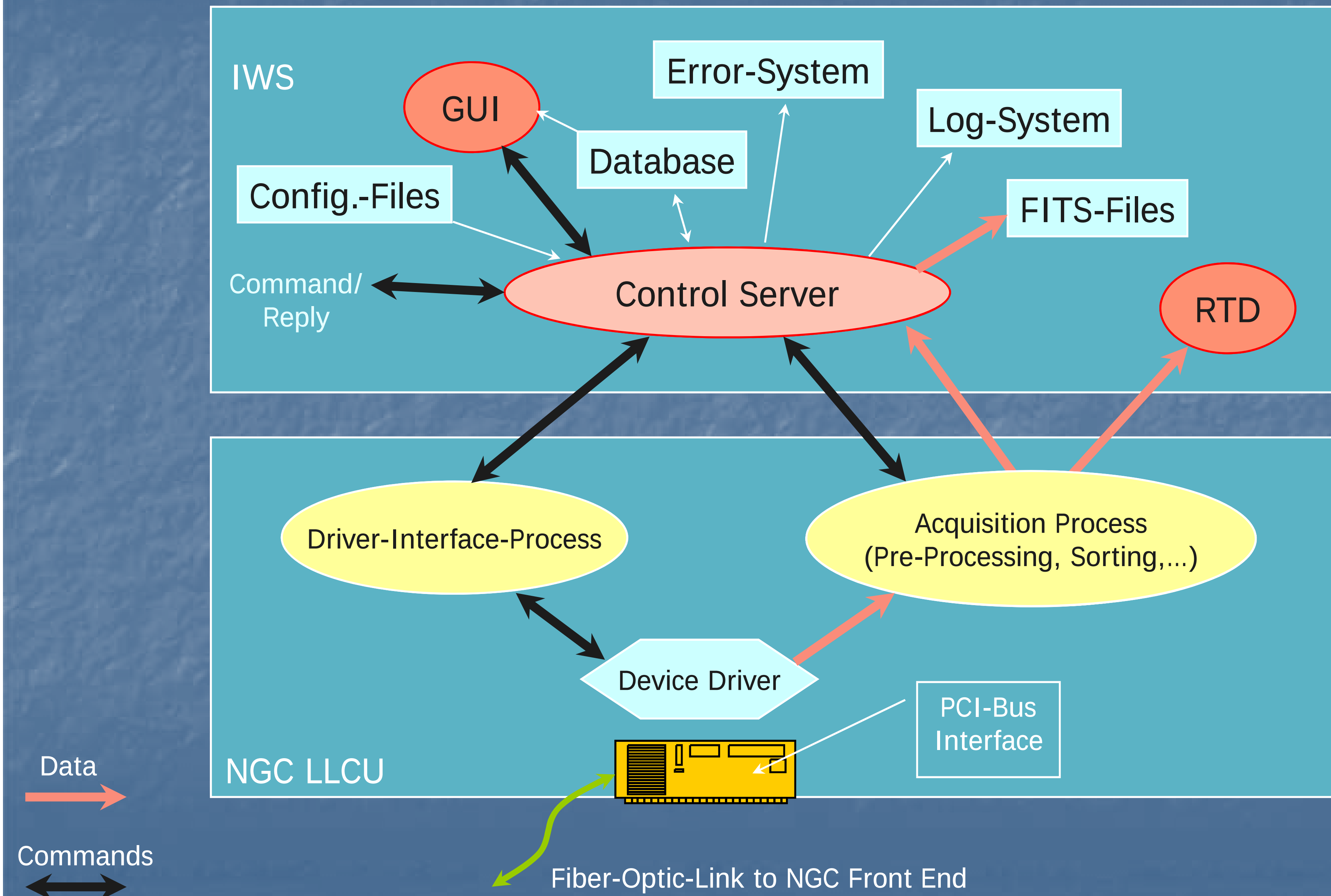


# Computing Architecture

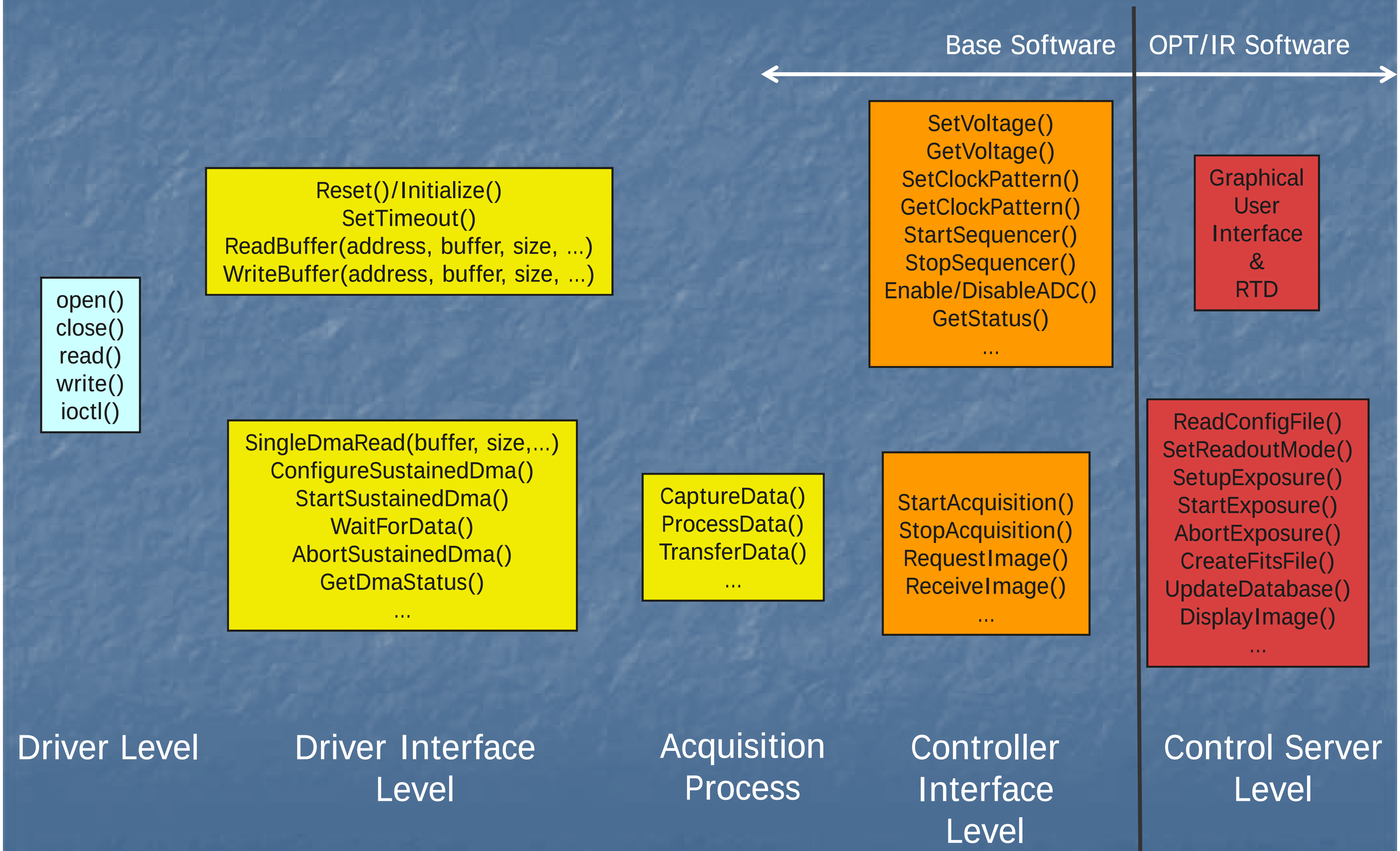



With the current Linux-PC model
we can achieve 200 Mbytes/s
sustained input data-rate with co-adding
(double correlated read-out)

# The Processes



# Software Hierarchy



# Software Modules

- *dicNGC* - Dictionary (both OPT/IR)
- *ngcdrv* - Device Driver
- *ngcb* - Driver Interface and Basic Routines
- *ngcpp* - Pre-Processing
- *ngcdcs* - Control Software & Server

Base SW

- *ngcgui* - Engineering & IR GUI
- *ngcrtd* - Engineering & IR Real-Time Display

IR SW +
Opt. SW (engineering)

- *ngciracq* - IR Acquisition Processes
- *ngcircon* - IR Control SW & Server
- *ngclcu* - NGC-LCU Interface SW (IR, for VLTI)

IR SW

205726 lines of code

- The modules will be part of the VLTSW Releases.
- All modules contain Test Procedures for TAT (automated testing).

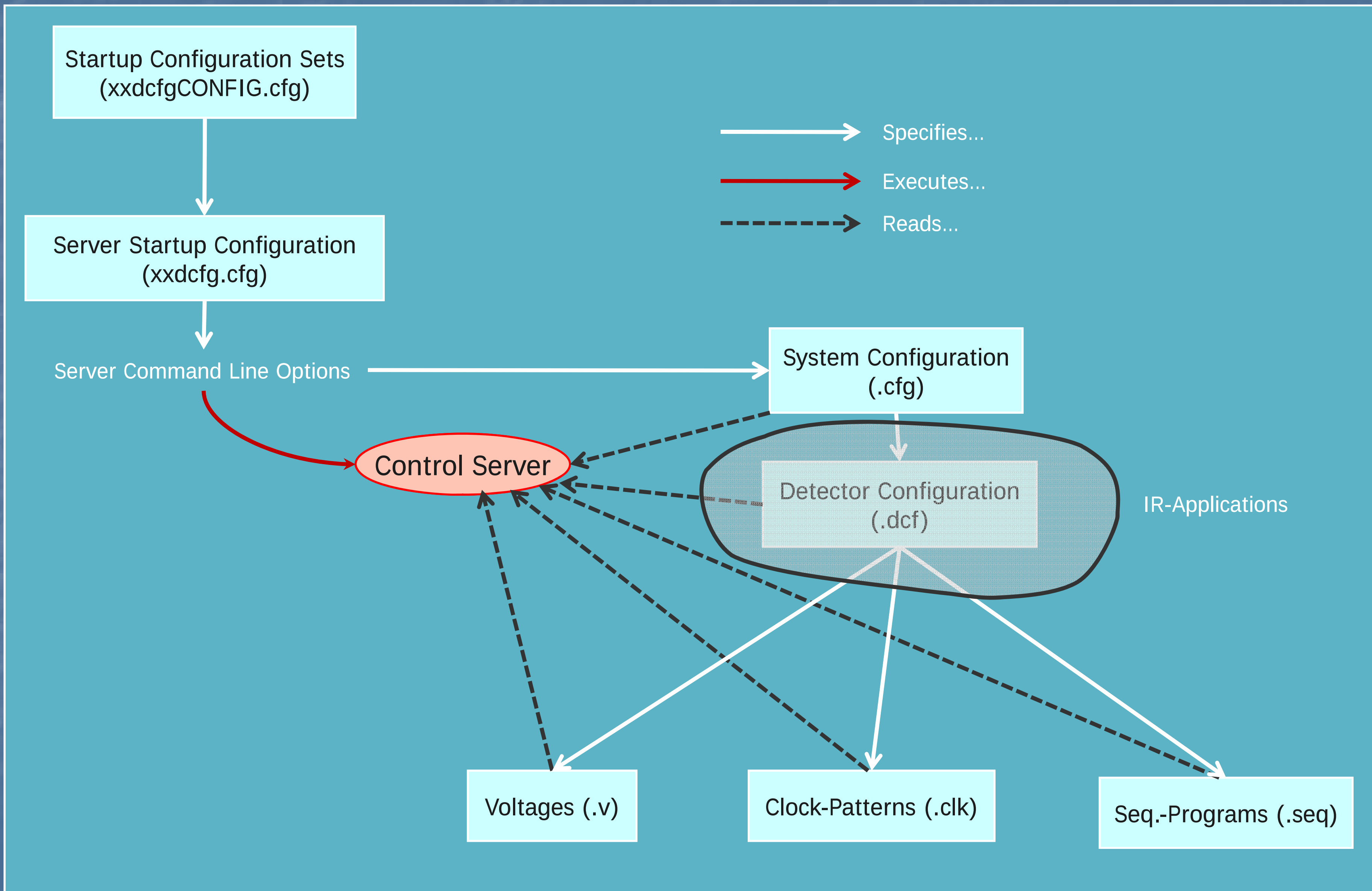
# Installation Procedure IWS and NGC LLCU

- Via installation scripts:
  - cmmCopy ngcarch
  - cd ngcarch/src/
  - make all install (fixed versions)
  - make update install (latest versions)
- ngcins software module contains a pkgin installation-configuration (for both NGC IR and OPT software).

# Installation Procedure
# Device Driver

- Retrieve the driver module from the archive (if not yet done):
  - cmmCopy ngcdrv
  - cp –r ngcdrv /tmp
- Login as “root” to continue the installation (Attention: use telnet or “su –” to ensure proper root session):
  - cd /tmp/ngcdrv/src
  - make all install
- Now you can load/unload the driver with:
  - /usr/local/bin/ngcdrv_load
  - /usr/local/bin/ngcdrv_unload
- Add the following line to the file “/etc/rc.local” to load the driver at boot-time:
  - /usr/local/bin/ngcdrv_load
- The device driver creates two independent devices:
  - “/dev/ngc<i>_com” and “/dev/ngc<i>_dma”.

# Configuration Files Overview



# Startup Procedure

- Startup tools:
  - ngcdcsStartServer <configuration-set name> [-gui] [other options]
  - ngcdcsStopServer <configuration-set name>
  - ngcdcsStartGui <configuration-set name> (only GUI)
- [other options] for maintenance (verbose mode etc.) or for overriding the keywords defined in the Startup-Configuration-Set.
- The control server startup configuration is described by a unique name (configuration-set name).
- The <name> refers to the name of a Startup-Configuration-Set which is defined in the main configuration file:
  - $INS_ROOT/SYSTEM/COMMON/CONFIGFILES/xxdcfgCONFIG.cfg
- The Startup-Configuration-Set describes the Startup-Configuration-File and some administrative options:
  - CONFIG.SET1.NAME "KMDCS";
  - CONFIG.SET1.DICT "NGCCON";
  - CONFIG.SET1.FILE1 "kmdcfg.cfg";
  - CONFIG.SET1.PERM1 664; # all
  - CONFIG.SET1.BACKUP T;
  - CONFIG.SET1.LOG T;
- The Startup-Configuration-File defines the server startup options - e.g. auto-online, auto-start, database-point (if not default), server instance and the controller electronics system configuration file (i.e. HW-configuration) to be loaded at startup.

# Startup Procedure

- Startup-Configuration-File:
  - # Control server name
  - DET.CON.SERVER  "ngcdcsEvh";
  - # Database point
  - DET.CON.DATABASE  "ngcdcs";
  - # Instance label for server and OLDB
  - DET.CON.INSTANCE  "";
  - # HW system configuration file
  - DET.CON.SYSCFG  "NGCIRSW/my_ngc.cfg";
  - # Startup mode (NORMAL, HW-SIM, LCU-SIM)
  - DET.CON.DFEMODE  "HW-SIM";
  - # Go online after start
  - DET.CON.AUTONLIN  F;
  - # Auto-start at online
  - DET.CON.AUTOSTRT  F;
  - # Enable sub-system status polling
  - DET.CON.POLL  T;
  - # Detector system index (DETi.XXX)
  - DET.CON.DETIDX  1;
  - # Dictionaries to load for this detector system
  - DET.CON.DICT  "NGCDCS";
  - # GUI name
  - DET.CON.GUI   "ngcgui";

# NGC System Configuration

- The System-Configuration-File describes the physical NGC system architecture.
- Defines the interfaces (PCI-boards).
- Defines the Sequencer-, CLDC-, ADC- and Shutter-modules in the system.
- Defines the default setup for all modules (e.g. number of clocks, auto-enable, ADC-operation mode, ...).

# NGC System Configuration (Example)

```
# Device description
DET.DEV1.NAME    "/dev/ngc0_com";     # associated device name
DET.DEV1.HOST    "$HOST";             # host where interface resides
DET.DEV1.ENV     "$RTAPENV";          # server environment name

DET.DEV2.NAME    "/dev/ngc1_com";     # associated device name
DET.DEV2.HOST    "$HOST";             # host where interface resides
DET.DEV2.ENV     "$RTAPENV";          # server environment name

# CLDC modules
DET.CLDC1.DEVIDX    1;                # associated device index
DET.CLDC1.ROUTE     "2";              # route to module
DET.CLDC1.AUTOENA   "T";              # auto-enable at online
DET.CLDC1.MARGIN    0.2;              # margin for voltage check (in volts)
DET.CLDC1.DCGN      2.0;              # bias gain
DET.CLDC1.CLKGN     1.0;              # clock gain

DET.CLDC2.DEVIDX    2;                # associated device index
DET.CLDC2.ROUTE     "2";              # route to module
DET.CLDC2.AUTOENA   "T";              # auto-enable at online

# Sequencers
DET.SEQ1.DEVIDX     1;                # associated device index
DET.SEQ1.ROUTE      "2";              # route to module

# ADC modules
DET.ADC1.DEVIDX     1;                # associated device index
DET.ADC1.ROUTE      "2";              # route to module
DET.ADC1.NUM        4;                # number of enabled ADC units on board
DET.ADC1.BITPIX    16;                # number of bits per pixel
DET.ADC1.FIRST      "T";              # first in chain
DET.ADC1.PKTCNT     1;                # packet routing length (# of packets from down-link)

DET.ADC2.DEVIDX     1;                # associated device index
DET.ADC2.ROUTE      "5,2";            # route to module
DET.ADC2.NUM       32;                # number of enabled ADC units on board
DET.ADC2.BITPIX    16;                # number of bits per pixel
DET.ADC2.FIRST      "F";              # first in chain
DET.ADC2.PKTCNT     0;                # packet routing length (# of packets from down-link)
```

# Controller Programming

- The Clock-Patterns can be defined both in ASCII-Format (*xxx.clk*, *IRACE-style*) and in a new Binary Format (*xxx.bclk*, output of the Graphical Editing Tool BlueWave). The formats can be converted automatically.
- Synchronization with external events (e.g. trigger) can be done after any state in any clock-pattern.
- A new Sequencer Programming Language has been defined to make maximum use of the new HW capabilities (all code is executed at the same speed-level within the firmware). File extension is "*xxx.seq*".
- Multiple Sequencer Instances within one system are supported.
- The detector voltages are defined in a Voltage Configuration File in Short-FITS format (*xxx.v*).
- The voltage configuration files can be loaded to any CLDC instance in the system.

# Clock-Pattern Generation

```
# Clock mapping (can be spread over several lines).
# This maps the clocks described below onto physical clock lines.
# Mechanism is: Phys. clock line for logical clock n = MAP[n].
DET.CLK.MAP1  "1,2,3,33";   # Mapping list
DET.CLK.MAP2  "37,4";       # Mapping list

# Clock pattern definitions
DET.PAT1.NAME  “FrameStart";
DET.PAT1.NSTAT 5;
DET.PAT1.CLK1  "00000";
DET.PAT1.CLK2  "00000";
DET.PAT1.CLK3  "00000";
DET.PAT1.CLK4  "00000";      # Convert
DET.PAT1.CLK5  "00110";      # Start pulse
DET.PAT1.CLK6  "00000";
DET.PAT1.DTV   "2,2,2,2,2";  # Dwell-Time vector
DET.PAT1.DTM   "0,0,0,0,0";  # Dwell-Time modification flags

DET.PAT2.NAME  “ReadPix";
DET.PAT2.NSTAT 6;
DET.PAT2.CLK1  "000111";
DET.PAT2.CLK2  “111000";
DET.PAT2.CLK3  "000000";
DET.PAT2.CLK4  "000010";      # Convert
DET.PAT2.CLK5  "000000";      # Start pulse
DET.PAT2.CLK6  "000000";
DET.PAT2.DTV   "5,5,5,5,5,5"; # Dwell-Time vector
DET.PAT2.DTM   "1,1,1,1,1,1"; # Dwell-Time modification flags

# Up to ngcdcsSEQ_MAX_PAT (=2048) clock patterns in this format...
```

# Sequencer Programs

- The sequencer program defines the order of execution of the defined clock patterns.
- The sequencer programs are fully driven by Setup Parameters (e.g. DET.DIT, DET.NDIT, window parameters, ...).
- Support of Arithmetic Expression Evaluation (TCL-syntax) to derive any program-loop parameter from the setup parameters and to compute attributes like exposure time estimations and minimum DIT.
- Support of Sub-Routines and Include-Files to minimize the code length.
- The program complexity can be scaled:
  - Simply do not "USE" any setup parameter.
  - Simply omit the "SCRIPT" part for arithmetic expression evaluation.

# Sequencer Program Example

```
# PATTERN DECLARATION
# PARAMETER DECLARATION
USE DET.NDIT DET.SEQ.DIT DET.DITDELAY DET.NDITSKIP
# SUBROUTINE DECLARATION
SUBRT RESET DELAY FRAME
#  EVALUATE
SCRIPT
if {$svar(DET.NDIT) <= 0} {
    set svar(DET.NDIT) 1
}
set tr [expr {$time_r(RESET) / 1000.0}]
set tf [expr {$time_r(FRAME) / 1000.0}]
set td [expr {$time_r(DELAY) / 1000.0}]
set svar(DET.SEQ.MINDIT) $tf
set t1 [expr {($svar(DET.NDIT) + $svar(DET.NDITSKIP))}]
set svar(delFac) [expr {($svar(DET.SEQ.DIT) - $tf) / $td}]
set svar(ditDelay) [expr {($svar(DET.DITDELAY) / $td)}]
if {$svar(delFac) < 0} {
    set svar(delFac) 0
    set svar(DET.SEQ.DIT) $svar(DET.SEQ.MINDIT)
}
set svar(DET.SEQ.EXPTIME) [expr {($t1 * ($tr + $svar(DET.DITDELAY) + $svar(DET.SEQ.DIT) + $tf))}]
SCRIPT_END
#  EXECUTE
LOOP INFINITE
  JSR RESET
  JSR DELAY $ditDelay
  JSR FRAME
  JSR DELAY $delFac
  JSR FRAME
END
RETURN
# SUBROUTINES
RESET:
INCLUDE "Hawaii2RGReset.seq"
DELAY:
INCLUDE "Hawaii2RGDelay.seq"
FRAME:
INCLUDE "Hawaii2RGFrame.seq"
```

# Sequencer Program Example

**Hawaii2RGFrame.seq:**

```
#PATTERN DECLARATION
FRAME_START = 1
ROW_START = 2
PIXEL = 3
RESET = 4
DELAY = 5
TRIGGER = 6
DUMMYPIXEL = 7
PIXELRESET = 8
VERTICALCLOCK = 9
EN_UNBUF_B = 10
MAINRESETB = 11

# PARAMETER DECLARATION

# EVALUATE

# EXECUTE (readout of full frame)
EXEC TRIGGER 1
EXEC MAINRESETB 1
EXEC EN_UNBUF_B 1
EXEC FRAME_START 1
LOOP 2048
    EXEC ROW_START 1
    EXEC DUMMYPIXEL 2
    EXEC PIXEL 64
    #EXEC DUMMYPIXEL 8
END
EXEC VERTICALCLOCK 1

RETURN
```

# Detector Voltage Setup

```
# Offsets:
DET.CLDC.CLKOFF 10.0;   # Global clock voltage offset
DET.CLDC.DCOFF  10.0;   # Global DC voltage offset

# Clock Voltages:
DET.CLDC.CLKHINM1   "clk1Hi";           # Name
DET.CLDC.CLKHI1      3.000;             # Setup value
DET.CLDC.CLKHIGN1    1.0;               # Gain (optional)
DET.CLDC.CLKHIRA1   "[-9.000, 9.000]";  # Allowed range

DET.CLDC.CLKLONM1   "clk1Lo";           # Name
DET.CLDC.CLKLO1      0.000;             # Setup value
DET.CLDC.CLKLOGN1    1.0;               # Gain (optional)
DET.CLDC.CLKLORA1   "[-9.000, 9.000]";  # Allowed range

# Up to 16 clock voltages like this ...

# DC Voltages:
DET.CLDC.DCNM1   "DC1";                 # Name
DET.CLDC.DC1      0.000;                # Setup value
DET.CLDC.DCGN1    1.0;                  # Gain (optional)
DET.CLDC.DCRA1   "[-9.000, 9.000]";     # Allowed range

# Up to 20 DC-voltages like this ...
```

# External Synchronization

- Synchronization points can be inserted at any place in any clock pattern executed by the sequencer program (i.e. set the “wait-for-trigger” bit in the particular state). When reaching such a point, the pattern execution is suspended after the dwell-time of this state until the arrival of an external trigger signal.
- Example:

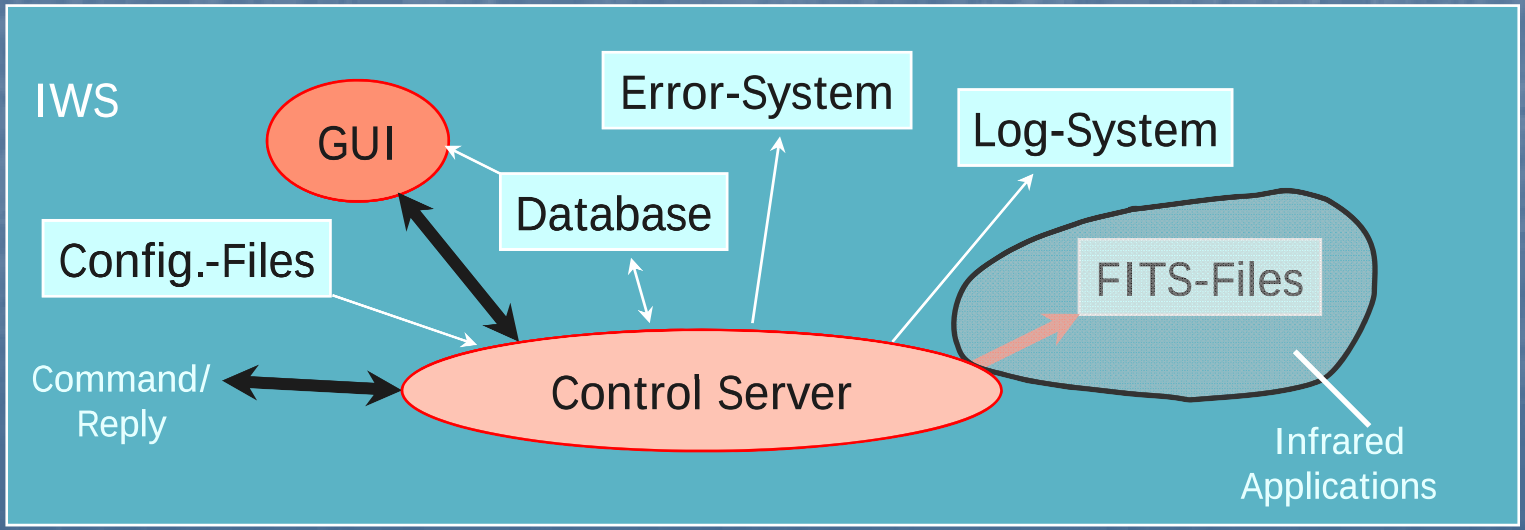
```
# Clock mapping (can be spread over several lines).
# This maps the clocks described below onto physical clock lines.
# Mechanism is: Phys. clock line for logical clock n = MAP[n].
DET.CLK.MAP1  "1,2,3,33";   # Mapping list
DET.CLK.MAP2  "37,4,61";    # Mapping list

# Clock pattern definitions
DET.PAT1.NAME  “FrameStartSync";
DET.PAT1.NSTAT 5;
DET.PAT1.CLK1  "00000";
DET.PAT1.CLK2  "00000";
DET.PAT1.CLK3  "00000";
DET.PAT1.CLK4  "00000";      # Convert
DET.PAT1.CLK5  "00110";      # Start pulse
DET.PAT1.CLK6  "00000";
DET.PAT1.CLK7  “10000";      # Sync
DET.PAT1.DTV   "2,2,2,2,2";  # Dwell-Time vector
DET.PAT1.DTM   "0,0,0,0,0";  # Dwell-Time modification flags
```

Wait for Trigger

# NGC-DCS Control Server

- The controller interface provides Modular Objects for Sequencer-, CLDC- and ADC-Control, for interfacing to the Acquisition Process and for the Asynchronous Data Reception (software module "*ngcdcs*").
- These objects can be assembled in the Control Server in an arbitrary way to reflect all functionality of any NGC hardware configuration (i.e. Multiple Instances of Sequencer-, CLDC-, ADC-modules and any number of Acquisition Processes). The module configuration is done through the System Configuration File.
- The control server can be used as NGC-HW Control Sub-System of the NGCOSW. That is the maximum degree of communality as the same compiled and linked object is used by both applications to access the HW. It can be configured at Run-Time for the one or the other purpose.



# Database

- The file ngcdcs.db contains the database branch definition for the control server. This file has to be included in the DATABASE.db file of the CCS environment.
- The following macros can be defined before each inclusion:
  - #define ngcdcsINSTANCE ngcdcs_myInstance
  - #define ngcdcsROOT :Appl_data:…:myPoint
  - #include "ngcdcs.db"
- The basic structure of the database is as follows:

```
--o <alias><ngcdcsINSTANCE >--    |--o system      (NGC system parameters)
                                  |--o exposure    (exposure parameters)
                                  |--o mode        (read-out mode parameters)
                                  |--o guiding     (guiding parameters)
                                  |--o chopper     (chopper interface)
                                  |--o seq_<i>     (sequencer parameters)
                                  |--o cldc_<i>    (CLDC parameters)
                                  |--o adc_<i>     (ADC module parameters)
                                  |--o acq_<i>     (acquisition module parameters)
```

- The branches for the Sequencer-, CLDC-, ADC-, and Acquisition- modules are indexed. One branch will be created per module.

# Database – Multiple Instances

- Define the instance in the Startup-Configuration-File:
  - # Control server name
  - DET.CON.SERVER "ngcdcsEvh";
  - # Database point
  - DET.CON.DATABASE "ngcdcs";
  - # Instance label for server and OLDB
  - DET.CON.INSTANCE "myInst";
- In the DATABASE.db file:
  - `#define ngcdcsINSTANCE ngcdcs_myInst`
  - `#define ngcdcsROOT :Appl_data:...:myPointForMyInstance`
  - `#include "ngcdcs.db"`
- You get the following interface:
  - Database: <alias>ngcdcs_myInst
  - Server Process: ngcdcsEvh_myInst

# Database

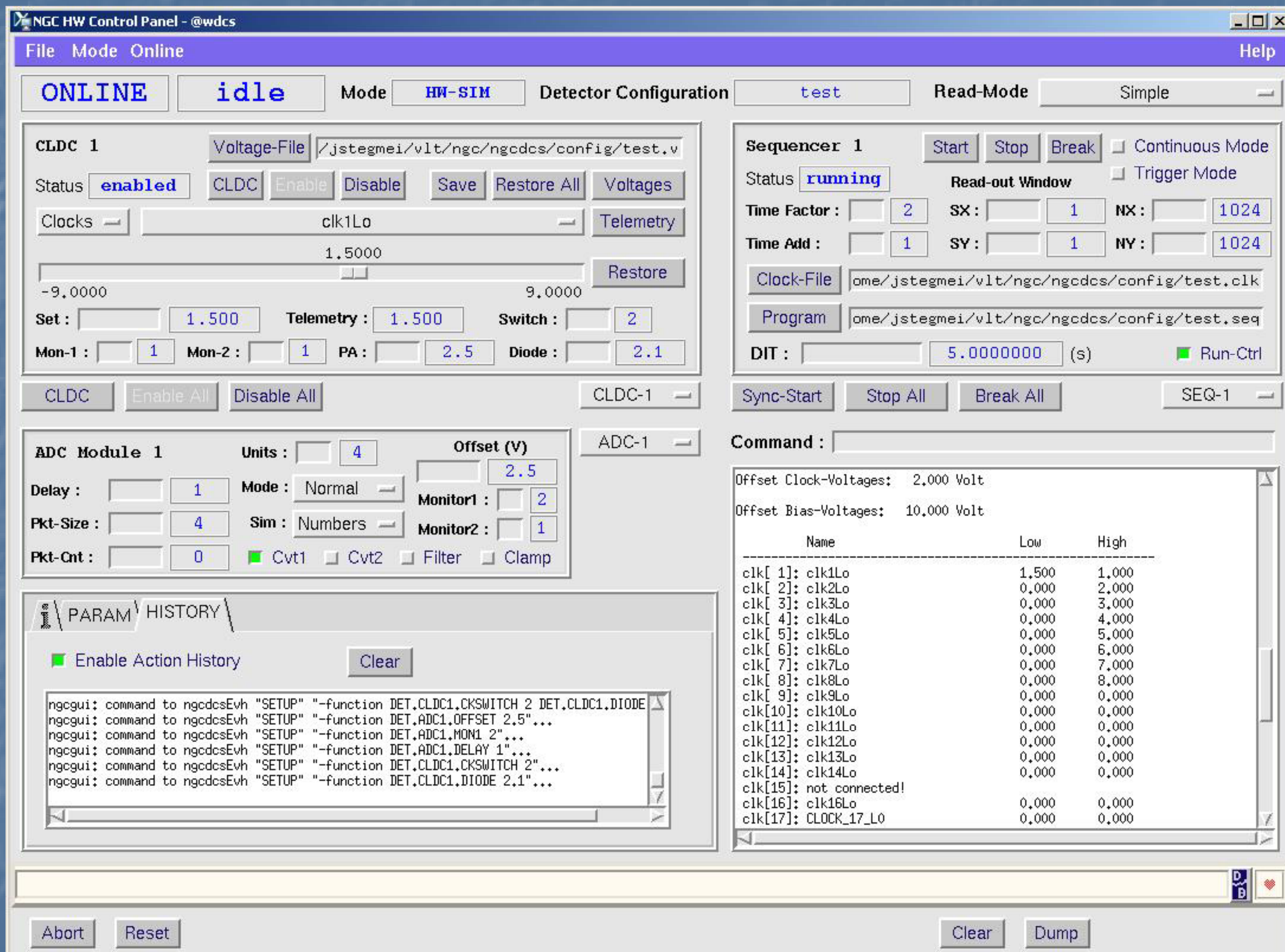
## Example for a 2 Camera-System

- xxdcfgCONFIG.cfg:
  - CONFIG.SET1.NAME "CAM1";
  - CONFIG.SET1.FILE1 "xxdcfg1.cfg";
  - CONFIG.SET2.NAME "CAM2";
  - CONFIG.SET2.FILE1 "xxdcfg2.cfg";
- xxdcfg1.cfg:
  - DET.CON.SERVER "ngcdcsEvh"; # -> *ngcdcsEvh_cam1*
  - DET.CON.DATABASE "ngcdcs"; # -> *<alias>ngcdcs_cam1*
  - DET.CON.INSTANCE "cam1";
  - DET.CON.SYSCFG "NGCIRSW/xxdcfgCam1.cfg"; # NGC HW-system configuration
- xxdcfg2.cfg:
  - DET.CON.SERVER "ngcdcsEvh"; # -> *ngcdcsEvh_cam2*
  - DET.CON.DATABASE "ngcdcs"; # -> *<alias>ngcdcs_cam2*
  - DET.CON.INSTANCE "cam2 ";
  - DET.CON.SYSCFG "NGCIRSW/xxdcfgCam2.cfg"; # NGC HW-system configuration
- In the DATABASE.db file:
  - `#define ngcdcsINSTANCE ngcdcs_cam1`
  - `#define ngcdcsROOT :Appl_data:...:CAMERA1`
  - `#include "ngcdcs.db"`
  - `#undef ngcdcsINSTANCE`
  - `#undef ngcdcsROOT`
  - `#define ngcdcsINSTANCE ngcdcs_cam2`
  - `#define ngcdcsROOT :Appl_data:...:CAMERA2`
  - `#include "ngcdcs.db"`

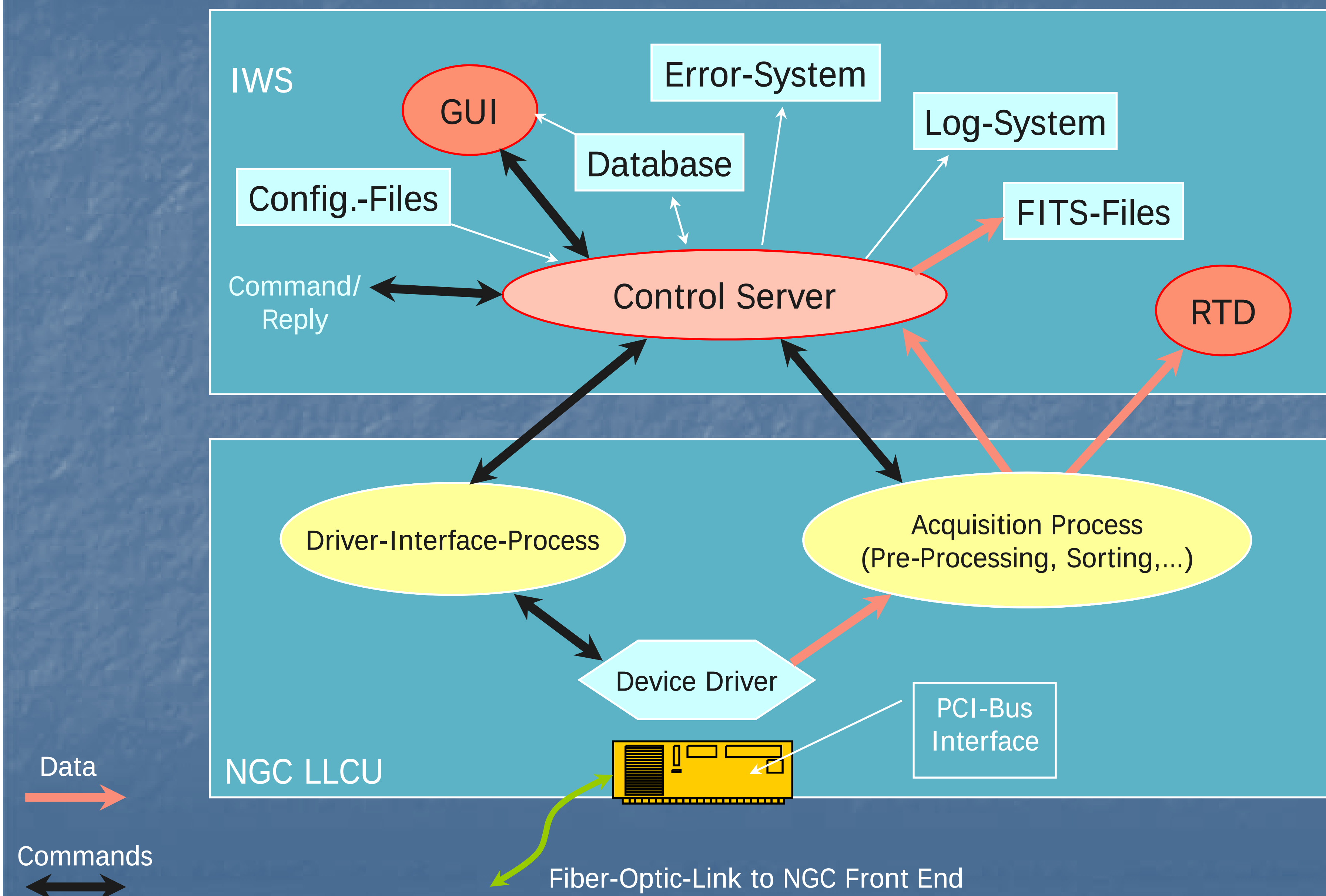
# Commands

- State Switching:
  - *STANDBY – ONLINE – OFF – EXIT*
  - *SIMULAT [-function HW|LCU]*
  - *STOPSIM*
- System Setup/Status:
  - *SETUP –function <parameter><value>*
  - *STATUS –function <parameter>*
- Exposure Control:
  - *START – WAIT* (wait until exposure completes)
  - *ABORT – WAIT* (wait until exposure is aborted)
- Hardware Control:
  - *SEQ -start, SEQ -stop*
  - *CLDC –enable, CLDC -disable*

# Graphical User Interface (ngcguiHw)



# NGC Infrared Software



# Data Acquisiton Processes

- The pre-processing framework for the multi-threaded Acquisition Process has been taken over from IRACE (software module "*ngcpp*").
- Currently this is required mainly for the data pre-processing in IR applications.
- Template Processes have been developed, which are an easy-to-use and stand-alone tool to visualize NGC raw-data on the RTD.
- The acquisition processes for the ESO Standard IR Detectors (HAWAII 1Kx1K, HAWAII2-RG 2Kx2K, ...) are assembled in a separate software module ("*ngciracq*"). Special setups (e.g. mosaics) for specific instruments may require special software modules ("*xxacq*").

# Frame Types

- User-definable Frame-Types (DIT, STDEV, HCYCLE, intermediate results...). The types can be selected to be generated and/or stored during an "exposure".
- Exposure Break-Conditions can be set per "per frame-type". This is the number of frames of a certain type to be stored during the exposure. The exposure terminates when all break-conditions are met. A zero value indicates to store as much as possible frames of that type until all other break-conditions are met.
- Individual SW-Windows per frame-type. A zero value for the dimension (*nx*, *ny*) indicates that the full frame will be requested from the acquisition process.

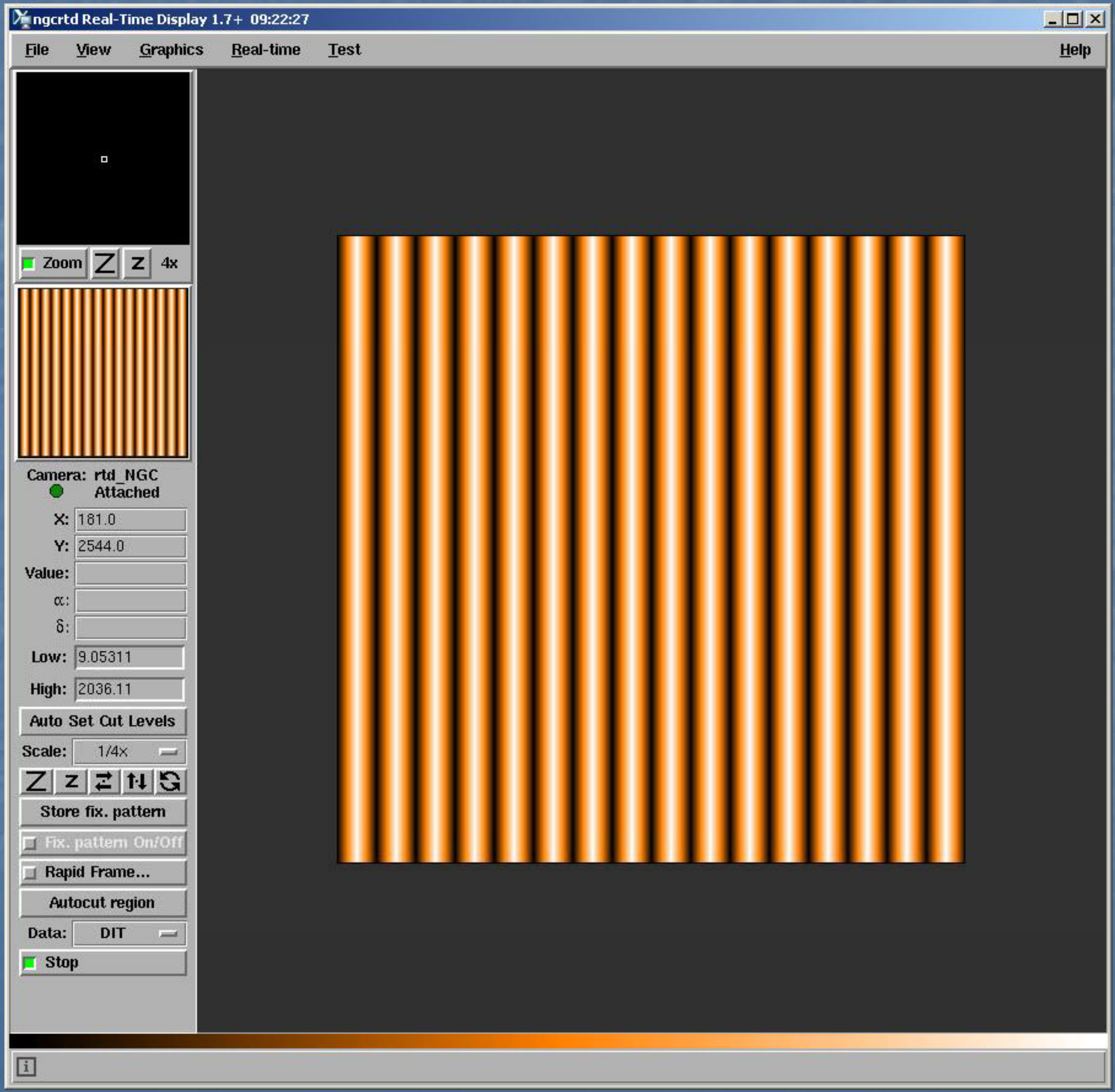
# Data Formats

- Default data format is “Binary Image Extension”.
- Data Cubes for Burst-Mode or for fast data acquisitions.
  - Minimum overhead
  - May require post-processing
  - One cube per frame-type
- Single files
  - For detector tests in the lab
  - To optimize merging process: start merging already before exposure is completed (e.g. VISTA-instrument).

# Data Interface

- FITS-Files
  - Wait for exposure termination and read the generated FITS-file(s).
- Direct connection to Acquisition Process (e.g. RTD)
  - Retrieve the binary image data with just minimum header information (dimension, type, sequential number).
- Post-Processing Call-Back
  - The control server calls a user-defined procedure before the frame is stored.

# NGC Real Time Display ("*ngcrtd*")



# Post-Processing Call-Back

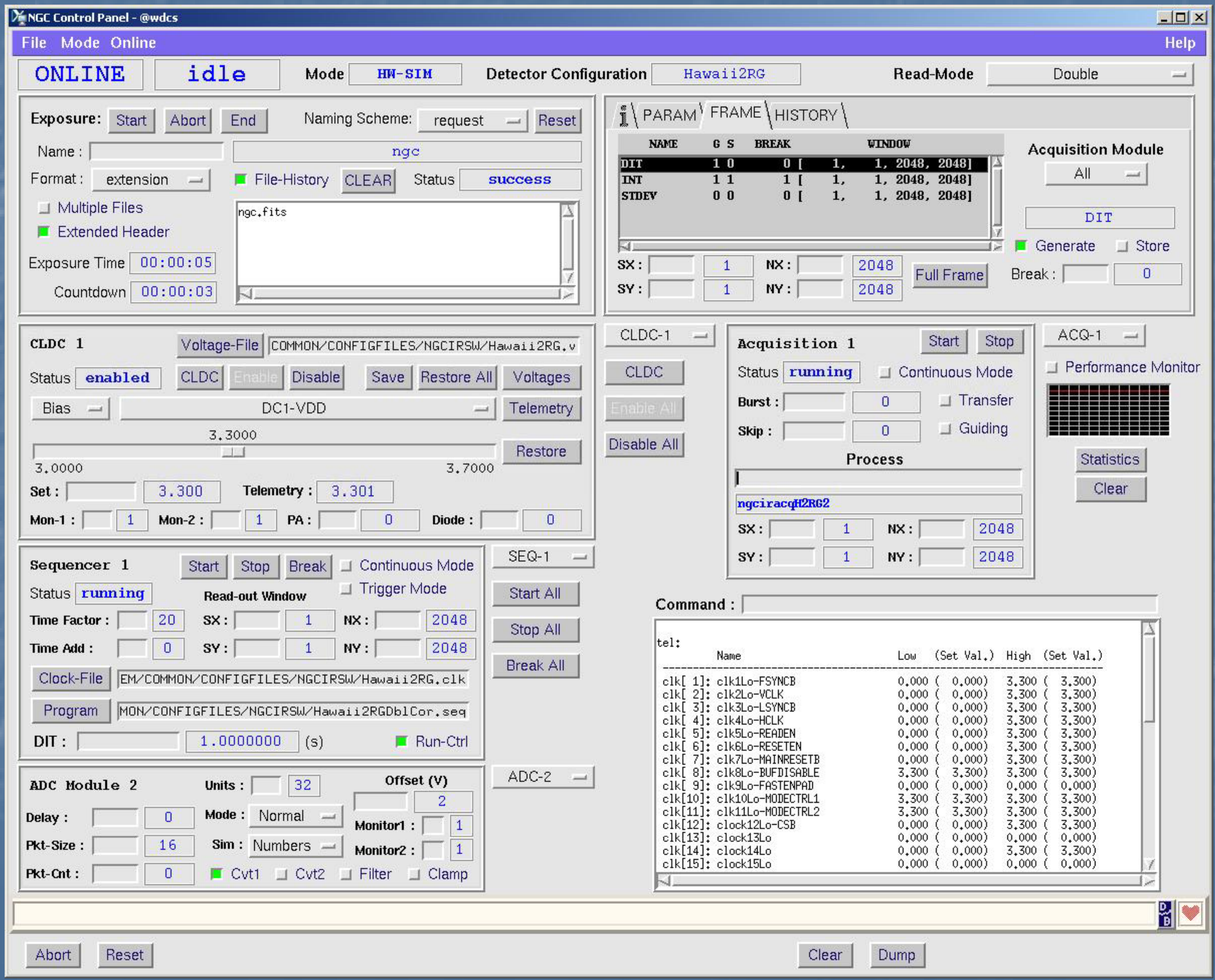
- The post-processing call-back is executed whenever a new data frame is received by the data acquisition thread of the control server:
  - int PostProcCB(void *buffer, ngcdcs_finfo_t *finfo, eccsERROR *error);

- The *ngcdcs_finfo_t* structure *finfo* contains all information for the *buffer*:

```
int type;          - Unique frame type
char name[64];     - Unique frame name
int fcnt;          - Frame counter
int scaleFactor;   - Scaling factor to be applied to normalize
int bitPix;        - Bits per pixel as defined in the FITS-standard
int sx;            - Lower left corner (x-direction)
int sy;            - Lower left corner (y-direction)
int nx;            - Dimension in x-direction
int ny;            - Dimension in y-direction
double crpix1;     - Reference pixel in x-direction
double crpix2;     - Reference pixel in y-direction
int detIdx;        - Detector index (for mosaics)
int expCnt;        - Exposure counter for this type
char utc[64];      - Time when frame was ready in the pre-processor
ngcdcsCUBE *cube; - Data cube object to be used for storing to a cube
```

- The post-processing call-back may return one of the following values:
  - ngcbSUCCESS - Successful operation
  - ngcbFAILURE - Failure (add an error string to the *error* stack)
  - ngcbSKIP - Successful operation - but skip all further actions on the frame (no storage to file,...)

# Graphical User Interface ("*ngcgui*")



# Application Specific Issues

- State Switching Call-Backs
  - The following call-backs are provided when the server state changes (i.e. upon reception of an *ONLINE*, *STANDBY* or *OFF* command):

```
ccsCOMPL_STAT OnlineCB1();
ccsCOMPL_STAT OnlineCB2();
ccsCOMPL_STAT StandbyCB1();
ccsCOMPL_STAT StandbyCB2();
ccsCOMPL_STAT OffCB1();
ccsCOMPL_STAT OffCB2();
```

  - The *xxxCB1()* functions are called before the state changes, the *xxxCB2()* functions are called after internal state switching.
- Setup/Status Call-Backs
  - The following call-backs are provided upon reception of a *SETUP* command:

```
ccsCOMPL_STAT SetupCB1(char **list, vltINT32 *size);
ccsCOMPL_STAT SetupCB2();
```

  - The following call-back is provided upon reception of a *STATUS* command:

```
int LookupCB(const char *name, char *value);
```

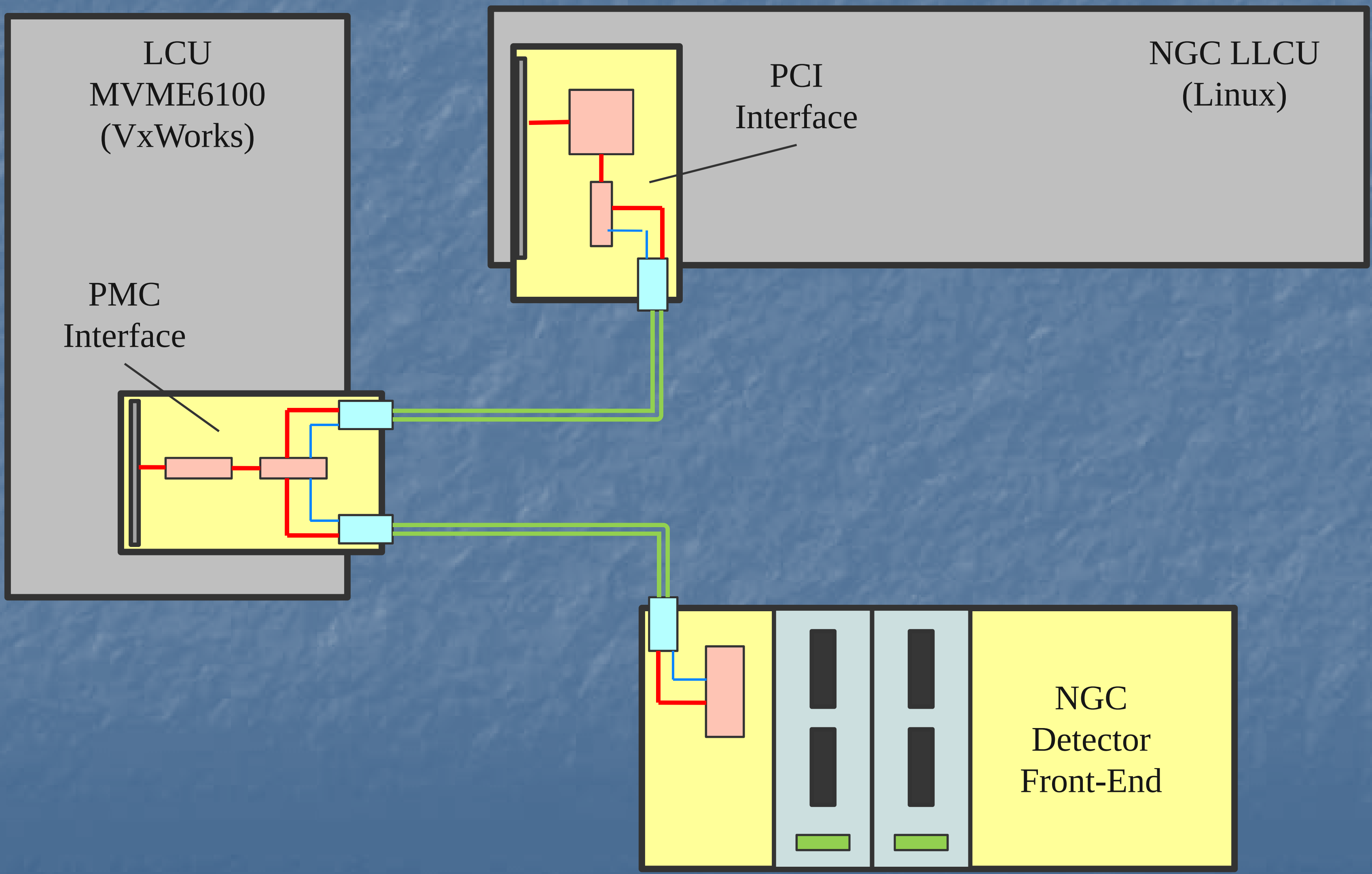
# Infrared Setup

- The data-taking is defined through "Read-Out Modes":
  - Read-out modes are defined by the Sequencer Program(s) running on the sequencer module(s) and by the corresponding Acquisiton Process(es) to be launched.
  - Read-out modes are selected by Name or a Unique ID (a Default Mode can be given).
- Window Read-Out is done by evaluating the window parameters within the sequencer program.
- The read-out modes and the voltage- and clock-pattern-configuration files to be loaded when going *ON-LINE* are defined in a Detector Configuration File. This also defines the detector parameters (size, type, name, mosaic arrangement, ...).

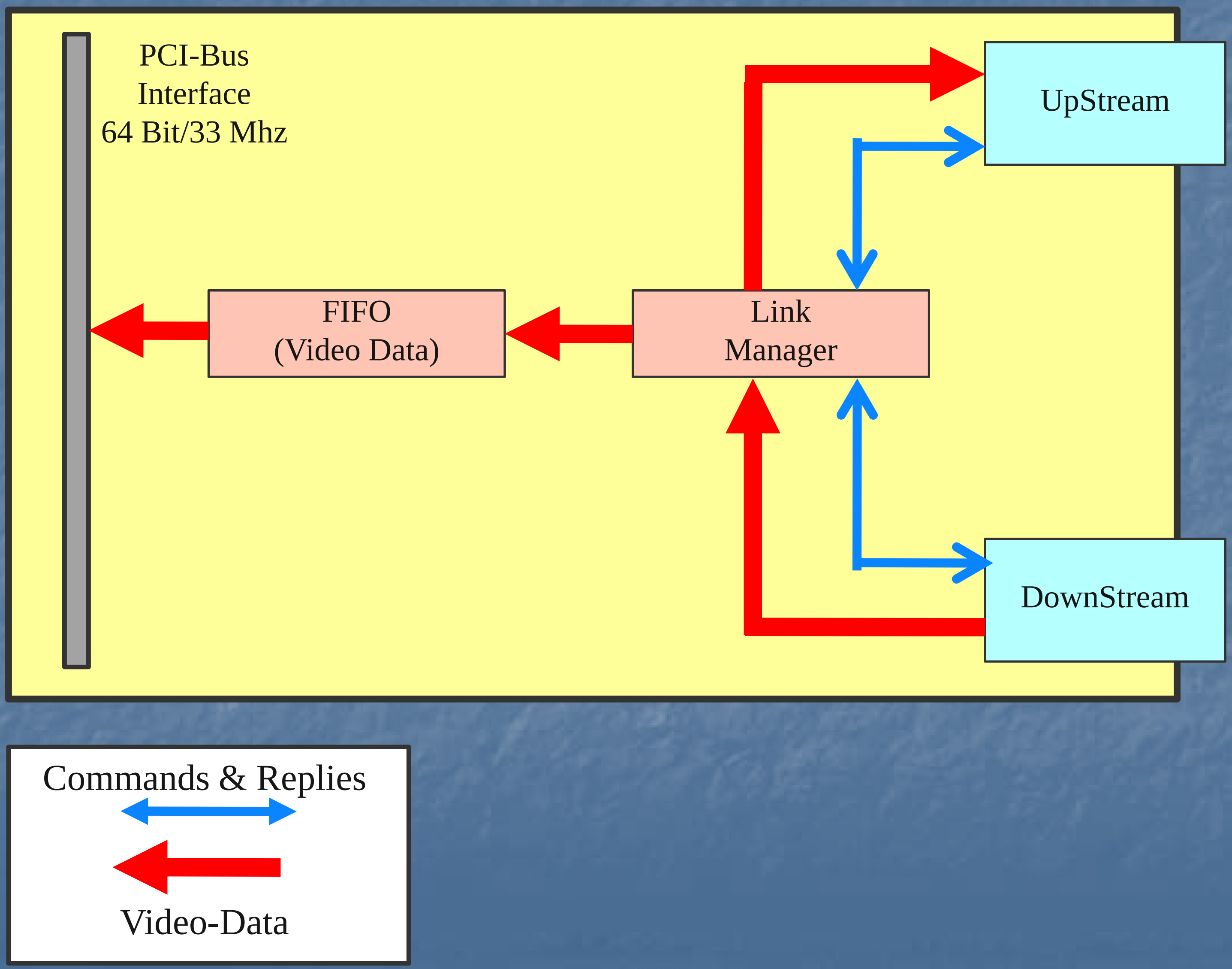
# Infrared “Exposures”

- Sustained Detector Read-Out and Video Display on the RTD (display remains active during the “Exposure”).
- Sustained Data-Transfer between NGC-LLCU and IWS for application specific Post-Processing (slow control loops, e.g. secondary auto-guiding).
- Starting an “Exposure” basically means “starting to transfer data to disk”.
- Burst-Mode for fast raw data acquisition.

# VLTI-System


LCU
MVME6100
(VxWorks)
PMC
Interface
PCI
Interface
NGC LLCU
(Linux)
NGC
Detector
Front-End

# PMC Interface (for VLTI)



# NGC-LCU Interface Software

- Software module "*ngclcu*".
- VxWorks Device Driver for the NGC PMC Interface card.
- Sustained DMA (64 Bit / 33 MHz, 128 MPixels/s)
- Data Capture Library
- Possibility to install a User-Defined Interrupt Service Routine (to minimize the latency).
- Latency: min. 4 µs, max. 6 µs depending on the configurable DMA-Blocksize (32 – 512 Bytes).
- Maintenance & Test Tools
  - Remote access from NGC-LLCU to board registers
  - Visualize data on RTD
  - Check data integrity

# Preview

- Integration into VLTSW-Release.
- New Detectors (Aquarius).
- Control SW for Sidecar ASIC.
- General procedure for Multiple Window Read-Out.
- Handling of the Guide-Window for the HAWAII2-RG array (parallel exposures).
- Acquisition processes for AO-Applications.

# Documentation

- VLT-MAN-ESO-13660-4510 NGC - User Manual
- VLT-MAN-ESO-13660-4085 NGC Infrared DCS - User Manual
- VLT-MAN-ESO-13660-4086 NGC Optical DCS - User Manual
- VLT-MAN-ESO-13660-4560 NGC-LCU Interface SW – User Manual
- VLT-LIS-ESO-13660-3907 NGC Project Glossary
- VLT-LIS-ESO-13660-3908 NGC Project Acronyms