SOFTWARE FOR THE NEW GENERAL DETECTOR CONTROLLER (NGC)

Jörg Stegmeier & Claudio Cumani

European Organisation for Astronomical Research in the Southern Hemisphere, Karl-Schwarzschild-Str. 2, 85748 Garching, Germany

Abstract: This article describes the architecture and the performances of the control software for NGC, the New General detector Controller under development at ESO. NGC is able to run a large number of infrared and optical detectors and will be used also for Adaptive Optics applications.

Key words: detector controllers, data acquisition, software

1. INTRODUCTION

NGC is the ESO New General detector Controller (*Baade et al. 2008* [1]), designed to handle the detectors of both optical and infrared instruments, for scientific imaging as well as advanced signal sensing applications. Basically the NGC electronics is the same for both infrared and optical applications. Nevertheless there are many differences concerning the usage of the controller and the data acquisition and data handling procedures. To cover both applications in an effective way and also to have a certain backwards compatibility with the predecessors *IRACE* (*Meyer et al. 1996* [3]) and *FIERA* (*Beletic et al. 1998* [4] and *Cumani et al. 1998* [5]) different software architectures have been chosen.

The following paragraph summarizes the main differences:

- **Detector Read-Out Schemes**
  For infrared applications the clock-pattern generation is running in an infinite loop and the detector is read-out/reset all the times. The optical detector is read-out just once at the end of an exposure.

- **Data Handling**
  The optical application delivers one frame at the end of the exposure and the only processing to be done is pixel sorting and possibly offset correction (if not yet done by the HW). The infrared data require some pre-processing depending on the read-out mode of the detector in use. The read-out modes, the pre-processing algorithms and the setup-parameters for these algorithms are manifold and require a very high degree of flexibility. The pre-processing task produces an arbitrary number of different result frame types, which all have to be transferred and/or displayed on demand. This also has an impact on the interface to the video display (section 8.3) because the frames to be displayed are not necessarily the same as the ones to be stored on disk. The latter is always the case for optical exposures.

- **Exposure Loops**
  For infrared applications *starting an exposure* basically means starting to transfer the acquired data to a FITS-file (i.e. the server has to attach to and keep step with a running procedure). The end-of-exposure condition is flexible and depends on both the requested frame types and on the number of frames of each type to be produced and stored. The optical exposure always terminates with the saving of the data, which are read at the end of the exposure and follows a much more rigid scheme ("*wiping*" – "*integrating*" –

"*reading-out*" – "*transferring*" – "*completed*"). This scheme implies an active intervention of the control server during the exposure like the application of new voltages in each state and the additional shutter-control, whereas the infrared control server mainly reacts passively on incoming data-frames once the exposure is started. So basically the demands on process concurrency are very different in both cases.

The goals were now to have one common software basis for both optical and infrared applications with maximum heritage of the strength of the two predecessors *FIERA* and *IRACE*. The system should be configurable for all possible realizations (i.e. number of channels, voltage drivers, array dimensions) thus requiring a strongly modular software architecture. The clock-pattern generation should be easily programmable and also support the synchronization with external events.

2. SYSTEM ARCHITECTURE

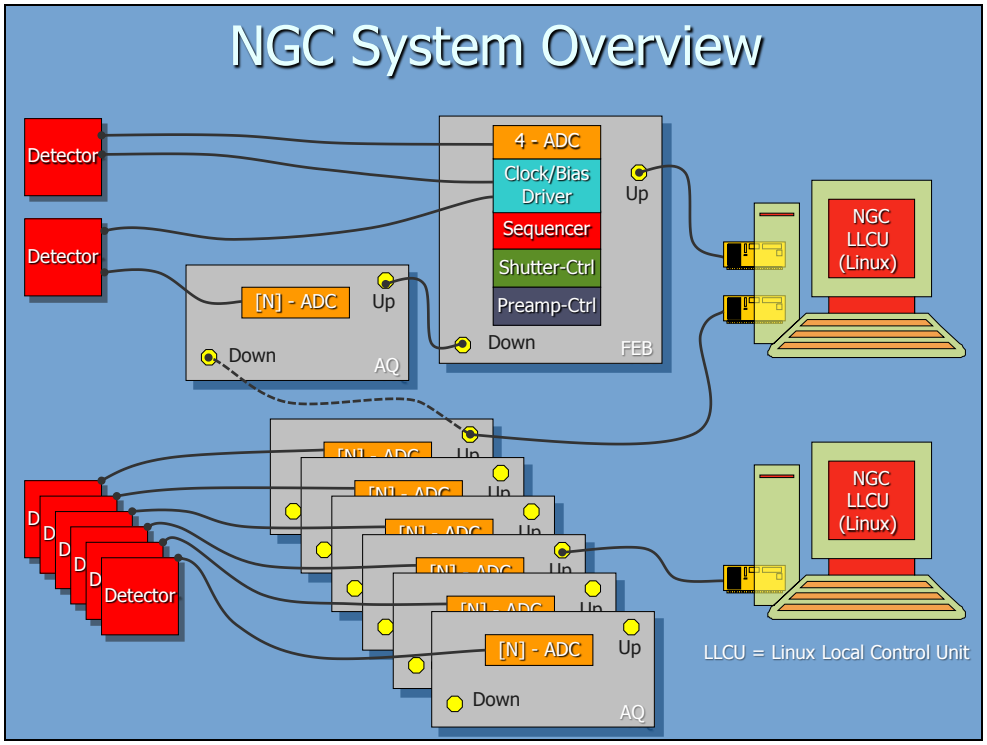


Figure 1 System Architecture

Figure 1 shows the overall system architecture with the maximum complexity. A detailed description of the hardware components is given in *Meyer et al. 2009* [2]. In the simplest case only one four-channel array is controlled via one front-end-basic board (FEB) connected to one hosting computer (NGC LLCU) via one PCI interface board. The NGC-LLCU is a PC running a Linux operating system (kernel 2.4 or higher). The PCI-board and the associated Linux device driver are both ESO developments. The front-end-basic board hosts four ADC components, the clock-pattern generator (sequencer), the clock- and bias-driver and auxiliary modules for shutter control and preamplifier control. When four channels are not enough additional acquisition boards (AQ) each hosting 32 ADC components may be added. When connecting through one single chain the bandwidth of the data link might be exceeded and the system needs to be accessed through additional PCI-Bus cards. This may imply the usage of more than one NGC-LLCU, also in case the computing or peripheral bus bandwidth of the NGC-LLCU is not sufficient. So the configuration range covers a simple system controlling one detector via one PCI-Bus interface card as well as large detector mosaics distributing their data via a huge number of channels among several computers.

3. THE PROCESSES

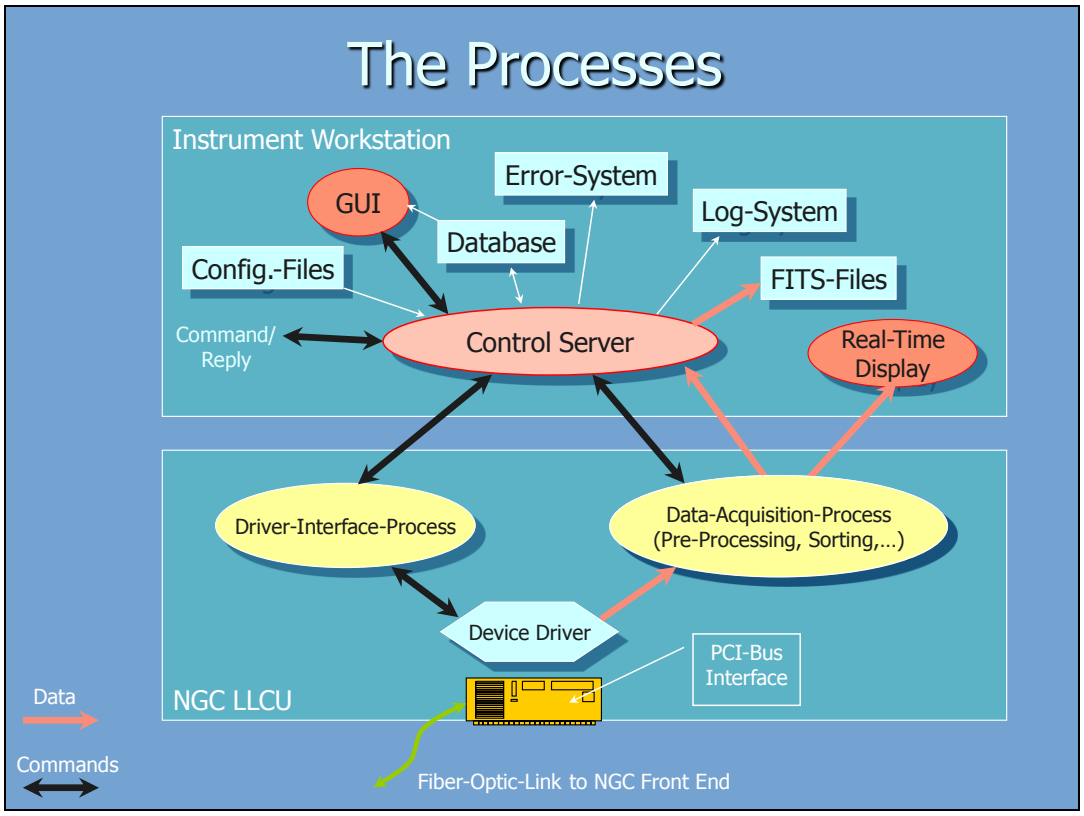


Figure 2 Processes

The NGC detector control software is running partly on the instrument workstation and partly on the NGC-LLCU, where the physical interface(s) to the NGC detector front end reside. **Figure 2** shows the processes running on this computing architecture. The control server communicates with the NGC electronics through driver interface processes running locally on the NGC-LLCUs. One driver interface process is launched by the control server per physical interface device. The acquisition processes (used for infrared data acquisition and pre-processing) are also launched and controlled by the control server. For maintenance and development operations all processes shown on the instrument workstation side may also run locally on one of the NGC-LLCUs. For software testing and software development all processes may run in simulation mode on the instrument workstation. All control software is coded in the C++ programming language. The graphical applications (see section 7) are based on TCL/TK. All software modules are under version control according to the ESO VLT-software standards which also employ automated testing facilities. The test procedures are launched every night and failures are reported automatically to the module owner. The NGC software is part of the yearly ESO VLT-software releases.

In order to optimize the commonality of the optical and infrared systems, the infrared detector control server can be scaled down to control only the NGC without doing any data acquisition or exposure handling. In this configuration it can simply be operated as a command driven sub-system of the NGC optical software. That is the maximum degree of commonality as the same compiled and linked object is used by both applications to access the controller electronics. The server can be configured at run-time for the one or the other purpose. It is also used as general engineering tool.

4. SYSTEM CONFIGURATION

The overall controller configuration is done through configuration files in short FITS format. For infrared applications it is divided into system configuration and detector configuration. The system configuration describes the physical NGC system architecture (e.g. number and addresses of boards in the system). It includes all information to identify the hardware configuration including the interface device names and the computing architecture (host names, environments, etc.). The detector configuration describes the usage of the system with respect to the connected detector(s) (i.e. which clock-patterns and which sequencer programs to load, which voltage setup to apply, etc.). There are cases, where more than one detector is driven by the same controller electronics and the switch between the detectors has to be done by applying a different detector configuration (i.e. enable/disable a different set of clock-drivers and/or ADC-modules). To reflect such cases, where different detector configurations are used on the same system configuration (or vice-versa the same detector configuration is used on different system configurations), the two files are kept separate in order to avoid unnecessarily duplicated information. Additionally a startup-configuration file (also in short FITS format) defines the command line options of the control server and is intended to be employed by higher level system startup tools. **Figure 3** shows the hierarchy of the NGC configuration files.

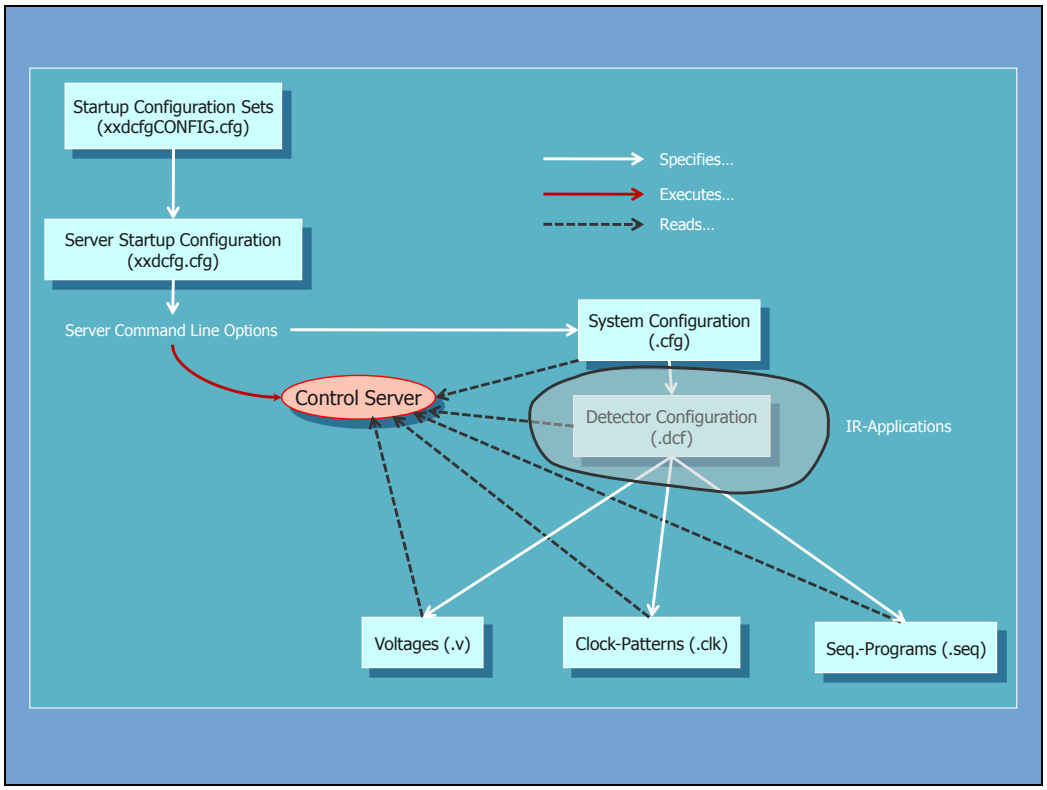


Figure 3 Configuration Files

5. CONTROLLER PROGRAMMING

The controller programming consists of the definition of clock patterns, sequencer programs and the voltage setup. The detector voltages are defined in a voltage configuration file in short FITS format. The voltages can be changed at run-time within a given range. Clock-pattern blocks can be defined both in ASCII-format and in a binary format, which is the output of the graphical editing tool *BlueWave* (section 8.5). The formats can be converted automatically. Synchronization with external events (e.g. trigger) can be done after any state in any clock-pattern block. A sequencer programming language has been defined to program the clock-pattern execution. There may be multiple sequencer instances within one detector front end system which can be operated synchronously based on one master clock.

5.1. Clock Pattern Generation

The clock pattern blocks define sequences of clock states which are stored in a RAM inside the NGC sequencer FPGA. The bits in the RAM define the state of each physical clock line plus some control bits ("wait-for-trigger", "end-of-pattern"). The duration of each state (dwell time) is defined in the state itself.

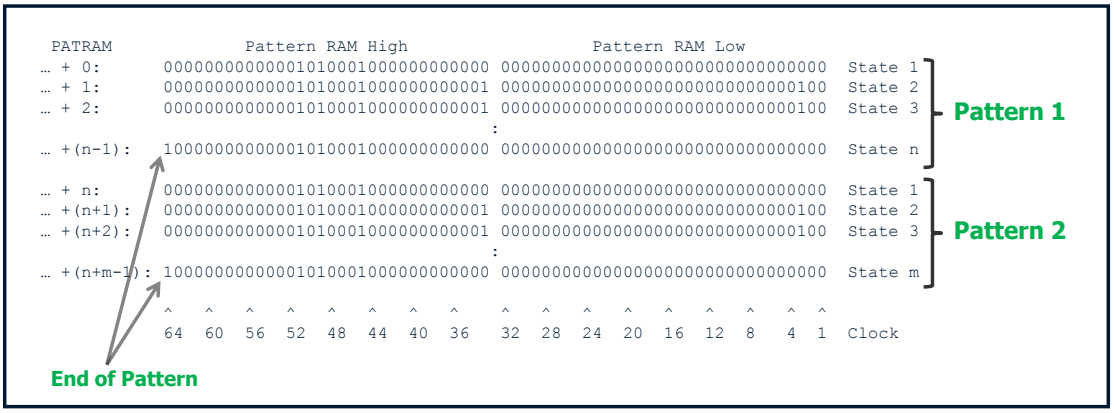
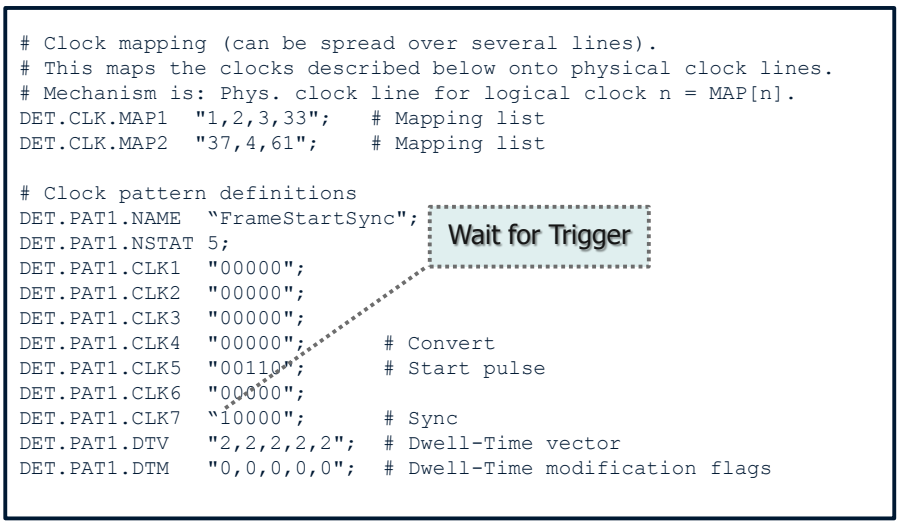


Figure 4 Clock Pattern RAM

The sequencer runs at a clock speed of 100 MHz (1 tick = 10 ns). So the dwell time of each clock state can be specified in steps of 10 ns. Synchronization points can be inserted at any place in any clock pattern by setting 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.




```
# 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
```

Figure 5 External Synchronization

5.2. Sequencer Programming

The sequencer program defines the order of execution of the defined clock pattern blocks. It is stored inside the sequencer FPGA in a simple 7 instruction code RAM.

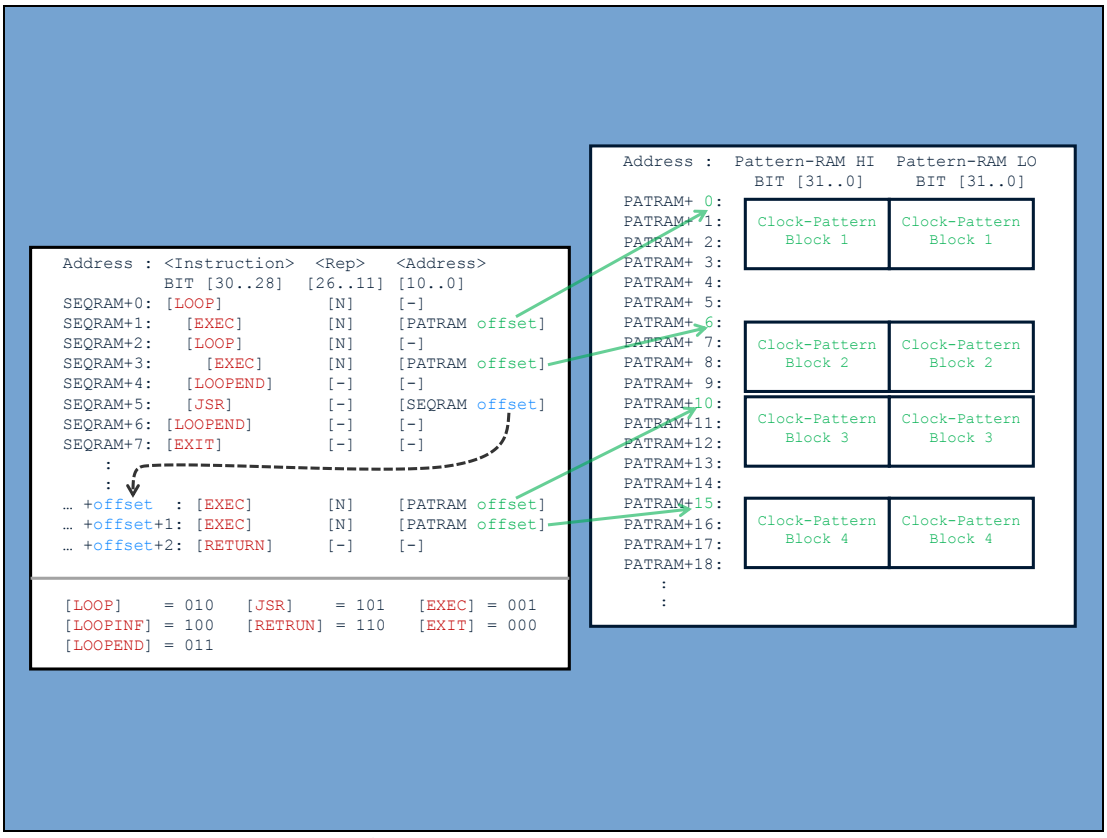


Figure 6 Sequencer Program RAM

The object code with the three basic commands *EXEC*, *LOOP* and *JSR* (jump-to-subroutine) can easily be compiled from a higher level programming language. This higher level sequencer program language is then fully driven by setup parameters (e.g. detector integration time, number of integrations, window parameters). The language supports arithmetic expression evaluation (TCL-syntax) to derive any program-loop parameter from the setup parameters and to compute attributes like exposure time estimation and minimum DIT. TCL has been chosen because it is a commonly used ESO software standard. The script evaluation may not be required by all applications. Where no expression evaluation is needed a simple code parsing can be done. Sub-routines and include files help to minimize the total code length.

6. INFRARED EXPOSURES

During an infrared exposure the data acquisition is usually running continuously and integrations are done one after another. Starting an exposure basically means to start transferring the data to a file on the instrument workstation. The display (see section 8.3) usually remains active during an exposure. Even when no exposure is running the software supports a sustained data-transfer between NGC-LLCU and instrument workstation in order to perform an application specific post-processing. This is needed for slow control loops (e.g. secondary auto-guiding).

6.1. Acquisition Process

The data acquisition process running on the NGC-LLCU is a multi-threaded pre-processing framework with a high task concurrency. Basically the pre-processing task to be performed is:

- Receive data from an external device. Generally this is achieved by writing the data from the device directly to the computer memory (DMA) and generating an interrupt once a certain amount of data had been transferred (data event). The data flow is continuous.
- Compute a data-array containing the result of the pre-processing - in the easiest case by sorting and/or adding up several subsequent input arrays. The computation ("adding-up", "sorting") must happen in parallel to the reception (DMA) of the next input data array. The algorithms are manifold and need specific parameters to be defined at run-time ("number of input arrays to add up", "number of frames to skip"). There are no strict real-time requirements (i.e. guaranteed response time on a data event), but the amount of data to be processed within a certain time-slot is huge (up to several hundred Mega-Bytes). The pre-processing must be done with higher priority than other system tasks which requires a priority based scheduling scheme (preemptive Linux kernel).
- Transfer the result array to disk and/or to a display. The target disk and the display utility usually reside on another computer (instrument workstation). Thus the transfer is done via a network interface. The transfer of the result array must also happen in parallel to both receiving and processing the next incoming data arrays (continuous data flow). Memory copies have to be avoided. This is a "low" priority task as the transfer only happens once after a certain amount of data has been processed.
- Handle asynchronous commands (e.g. "*SETUP*", "*START*", "*STOP*"). This has to be handled with highest priority to guaranty system addressability at all times.

The actual pre-processing algorithm is designed as a tiny main-process employing the threads framework. The thread scheduling and synchronization is fully transparent. There is one process per detector read-out mode which guarantees maximum independency and maximum safety as a new mode can never corrupt working code and all resources are released whenever the mode is switched.

Template processes have been developed, which are an easy-to-use and stand-alone tool to visualize NGC raw-data on the video display (see section 8.3). Standard acquisition processes for the ESO Standard IR Detectors (HAWAII 1Kx1K, HAWAII2-RG 2Kx2K, SELEX, AQUARIUS) are available within the NGC software package. Special setups (e.g. mosaics) may require special software modules.

6.2. Frame Types

The application specific acquisition processes may produce an arbitrary number of frame types ("*raw frame*", "*mean-value*", "*standard deviation*", "*chopper half-cycle frames*"). Each frame type has two flags associated to define whether frames of that type will actually be produced by the pre-processor and whether frames of that type should be stored to disk during an exposure. A software window can be defined individually for each type and for each acquisition module.

Usually an exposure is finished when the frame containing the mean value of several read-outs has been received on the instrument workstation. As it is required by some read-out modes to store also other frames during one exposure, a more general exposure break condition has to be applied: each frame generated by the acquisition process and selected to be stored can have a counter, which indicates the number of frames of that type to be stored during the exposure. The exposure is finished, when all of these frames have reached their *break condition*.

6.3. Burst Mode

In some cases it is necessary to store larger amounts of raw data or to sample at very high frame rates. If the frame rate is too high (> 200 Hz on most non-real-time UNIX platforms) the DMA-interrupt latency becomes dominating and no more CPU-power is left for pre-processing. Two kinds of "*burst modes*" are used to cover these two cases. One mode just fills up the memory with raw data and dumps it out to disk ("*raw data mode*"). The other mode internally increases the DMA-buffer and then does the normal pre-processing on several raw-data frames in one step in order to work around the frame rate limitation ("*internal burst mode*"). The "*internal burst mode*" mode is transparent to the pre-processing framework.

6.4. Data Interface

The standard data interface is FITS binary image extension format. The "*user*" waits for exposure termination and then reads the generated file. It is also possible to establish a direct connection to the acquisition process in order to retrieve the binary image data with just minimum header information (*dimension, type, sequential number*). This is normally done by displaying applications (video display). A third interface to the image data is the post-processing call-back. This is a user defined function which is called by the control server just before a frame is stored to disk. The function then may examine or modify the data and give it either back to the system or save it in its own format or to its own target (e.g. shared memory).

7. THE NGC SOFTWARE FOR THE OPTICAL DETECTORS

As described in the Introduction, optical and infrared detector controllers operate in different ways. For instance, optical detectors require an active intervention of the control server during the exposure (e.g. to apply new voltages when wiping, integrating or reading) or an interface to different kinds of shutters (for opening and closing, open/close delay times calculation, status checking).

As a consequence of the operational differences between infrared and optical instruments, at higher level the NGC software has to be divided into the two different branches, while the low level software - operating system and drivers - can be common to both infrared and optical applications thanks to the similarities of the detector controller hardware.

7.1. Finite state machine model

During the design phase of the NGC software for the optical instruments (NGC Optical Software, from now on NGCOSW), analysis has shown that detector controllers can be modeled as a ***finite state machine*** [6], i.e., by a model of behavior composed of a finite number of *states*, *transitions* between those states, and *actions* (activities that are to be performed at a given moment, e.g., when entering or exiting a state or during a transition).

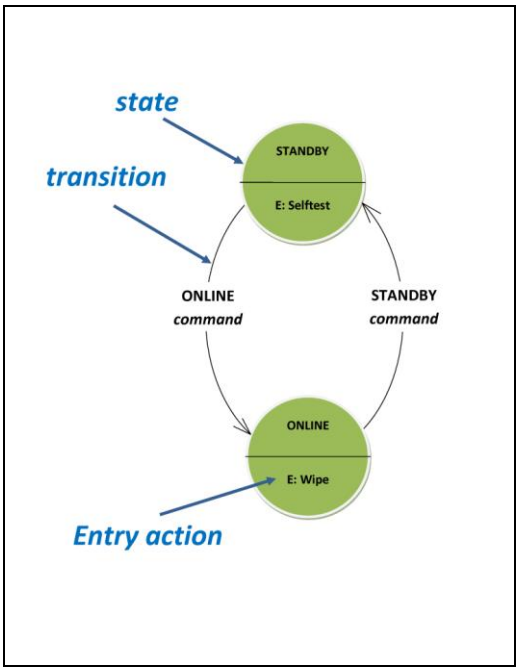


Figure 7 Example of a state machine

The usage of the finite state machine model has several advantages:

- it has a powerful ability to implement decision making algorithms;
- it is easy to create (it is basically defined by a table of possible states and relations among them, a "state machine configuration table");
- the design process involved in creating a state machine improves the overall design of the application;
- restructuring is very easy (it is basically just a redefinition of the state machine configuration table);
- it is the only model that allows "easy" code generation (meaning: the code implementing the state transitions).

The design has been implemented using UML (Unified Modeling Language) [7]. Finite state machines can be easily described in a graphical format by UML state charts (Figure 8 shows the UML state chart for a basic NGCOSW process, Figure 9 shows the complex UML state chart for the exposure coordination process).

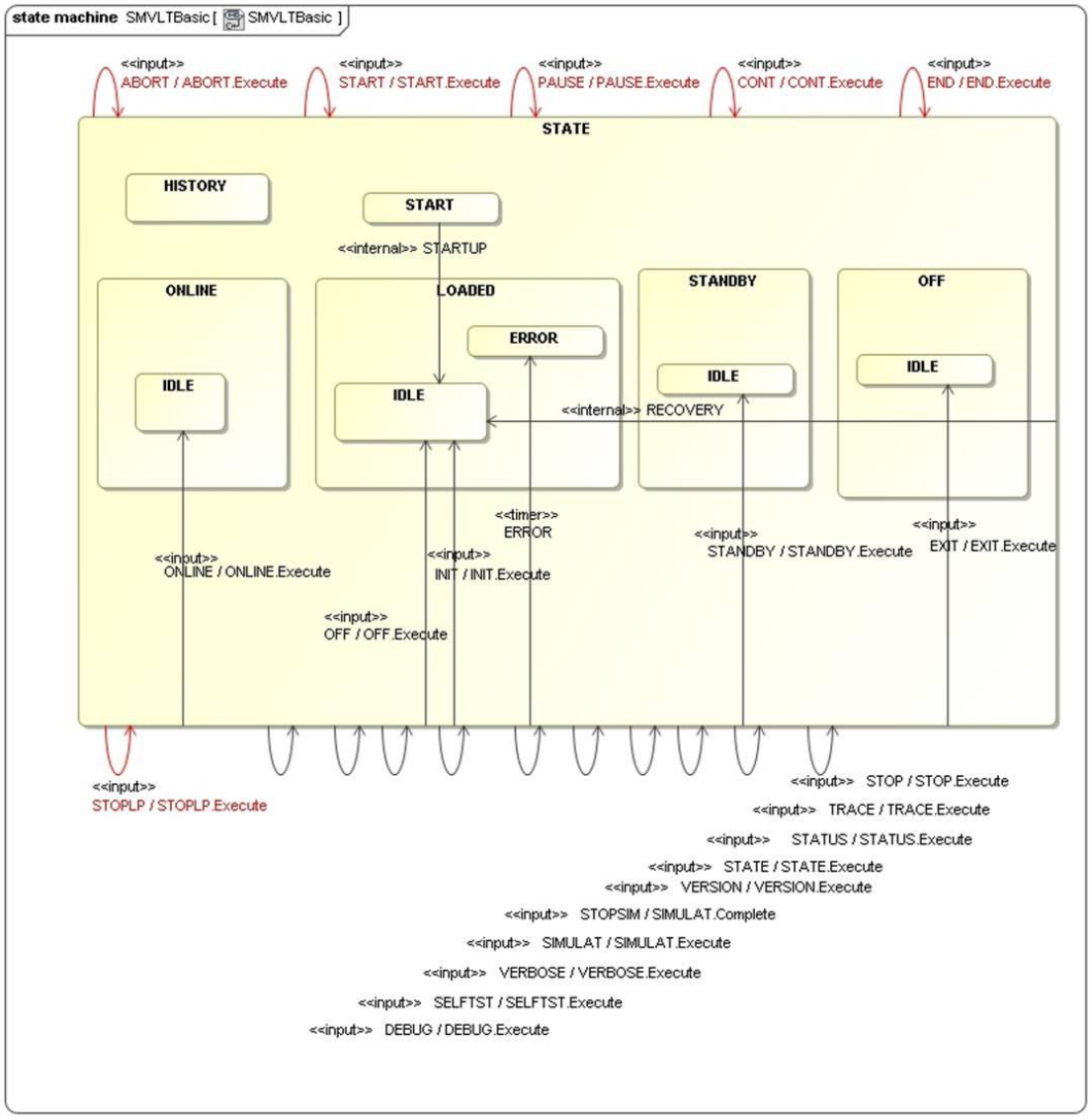


Figure 8 UML state chart for basic NGCOSW process

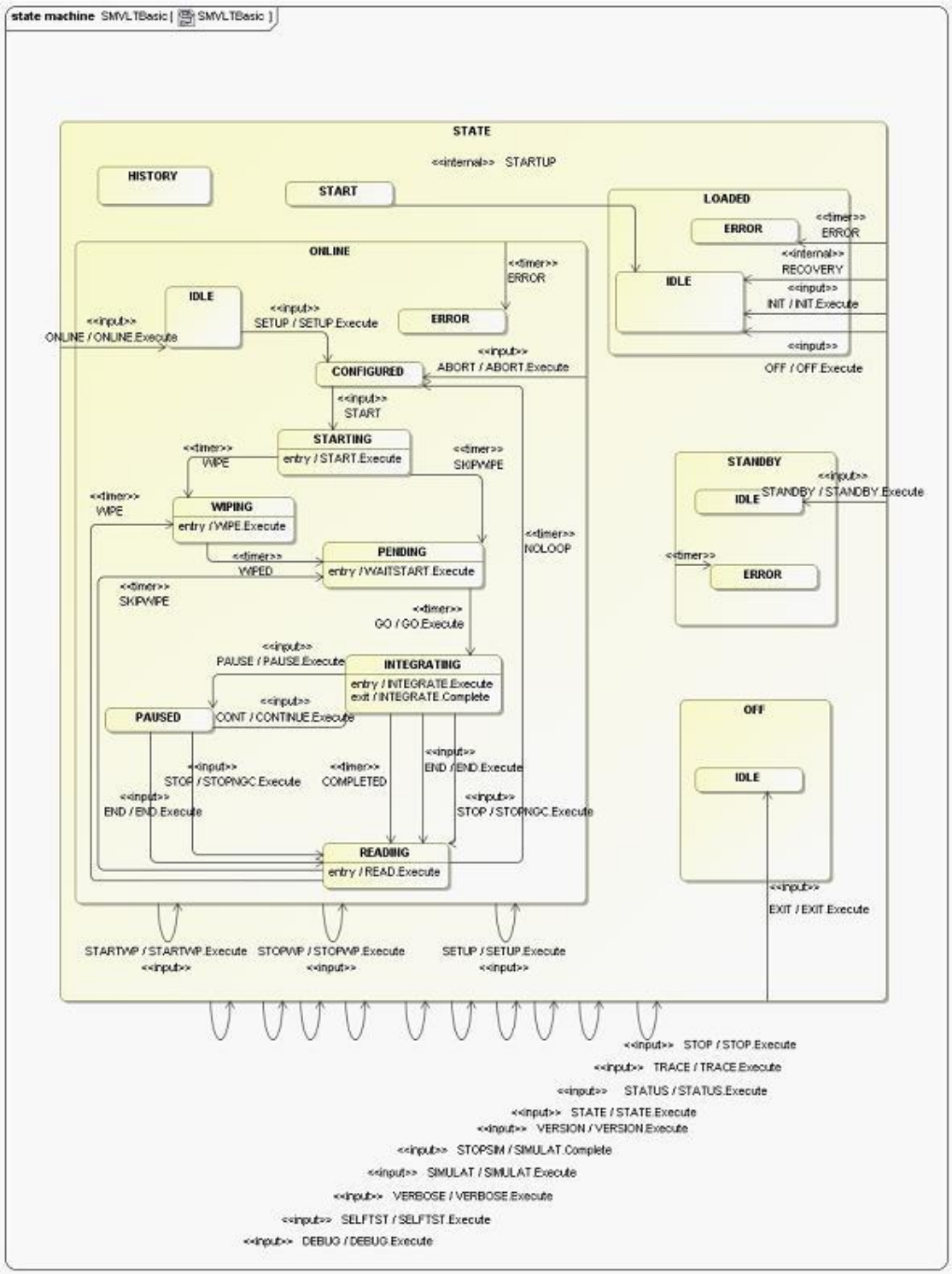


Figure 9 UML state chart for NGCOSW exposure coordination process

From a UML state chart it is possible to generate the state machine configuration table in an automatic way. Plug-ins for case tools like Enterprise Architect [8] and MagicDraw [9] have been developed at ESO for such a purpose.

The ESO **wsf** (workstation software framework) tool [10] has been used for the automatic generation of the code from the state machine configuration table. Note that wsf generates the code handling states, state transitions, messages, commands, error conditions, and so on, NOT the actions to be performed while moving between states (i.e., actions needed to drive an exposure). The implementation of these actions has to be performed on the generated code and depends on the characteristics of the detector, shutter, instrument requirements, etc.

7.2. Results

The amount of code generated for optical NGC consists of ca. 82.000 lines (27% of which is test software[1]) and corresponds to the code developed for FIERA, the previous optical detector controller produced by ESO.

The big difference in comparison with FIERA is that 78% of the NGCOSW has been automatically generated.

The whole NGCOSW design process has shown that the UML based modeling was an enormous improvement in terms of:

- *code robustness*: the finite state machine model forces a well structured design and can be "naturally" implemented in a test-driven development environment;
- *flexibility*: the finite state machine model can be easily modified in order to implement new requirements;
- *maintainability*: the code to be "manually" developed concerns only the actions required to drive a detector and is well confined (and defined);
- *development time*: the combined usage of UML and wsf has decreased the development time by generating a large portion of the code.

[1] For comparison: NGC Base Software consists of 220.000 lines of code (test software = 16%) and NGC IR-SW consists of 36.000 (test software = 12%)

8. GRAPHICAL USER INTERFACES

A number of graphical user interfaces have been developed to interact with the system.

8.1. NGC Hardware Control GUI

Figure 10 shows the basic panel to control just the NGC hardware. This does not contain any exposure control and is used as sub-application of the NGC optical software.

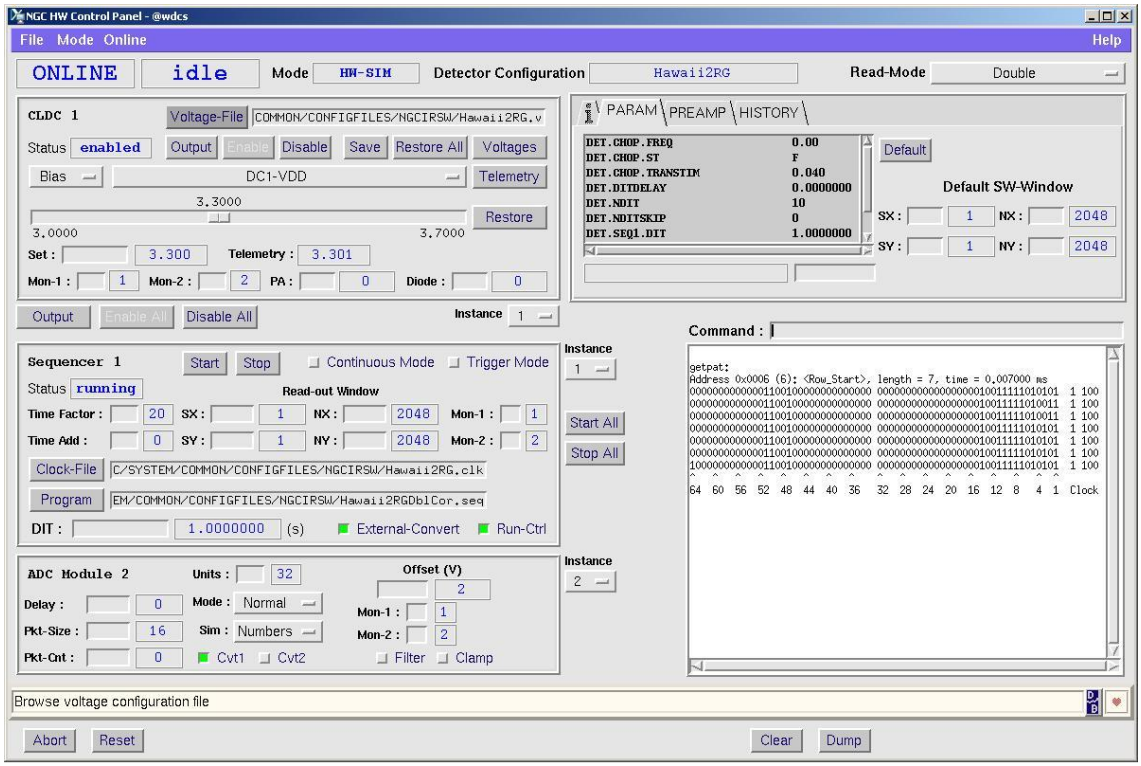


Figure 10 NGC Hardware Control GUI

8.2. Infrared Engineering GUI

Figure 11 shows the engineering GUI for infrared applications. This is also used as stand-alone tool for hardware development as by applying a default acquisition process “exposures” can be started and quick series of the raw-images plus mean-value and standard deviation can be saved to FITS-files for off-line evaluation.

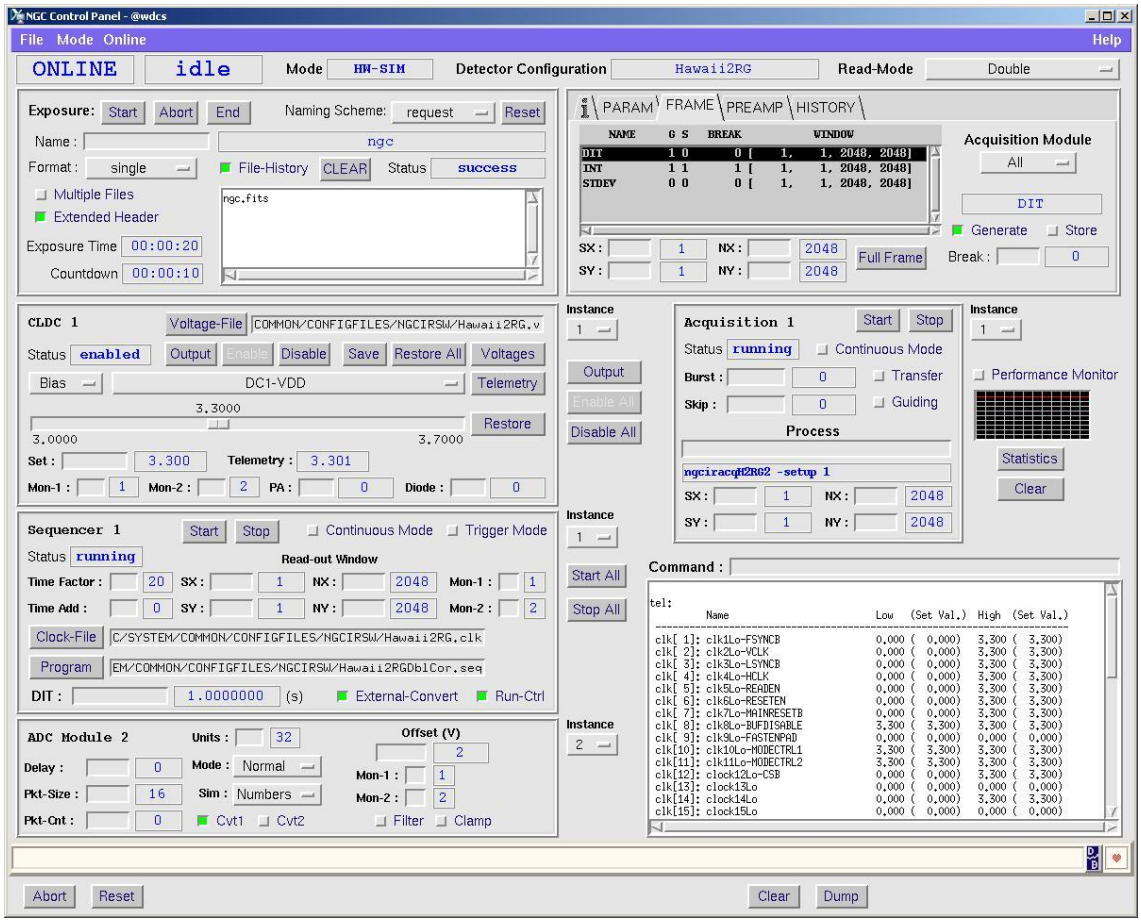


Figure 11 Infrared Engineering GUI

8.3. Optical Engineering GUI

Although the standard way to interact with NGCOSW is via VLTSW commands, a graphical user interface has been developed to easily operate the software in standalone mode (for instance, for testing in the lab).

Via the "System Status and Control" area the system can be started, put to standby and online, and terminated. System status is also monitored.

To perform an exposure, appropriate parameters like the exposure mode, time, type (normal, dark), etc must be set in the "Exposure Setup", and then the appropriate actions must be performed by using the options shown in the "Exposure Control" area.

By moving the cursor on an area or button, a message help concerning the pointed object is shown in the "help area" at the bottom of the panel.

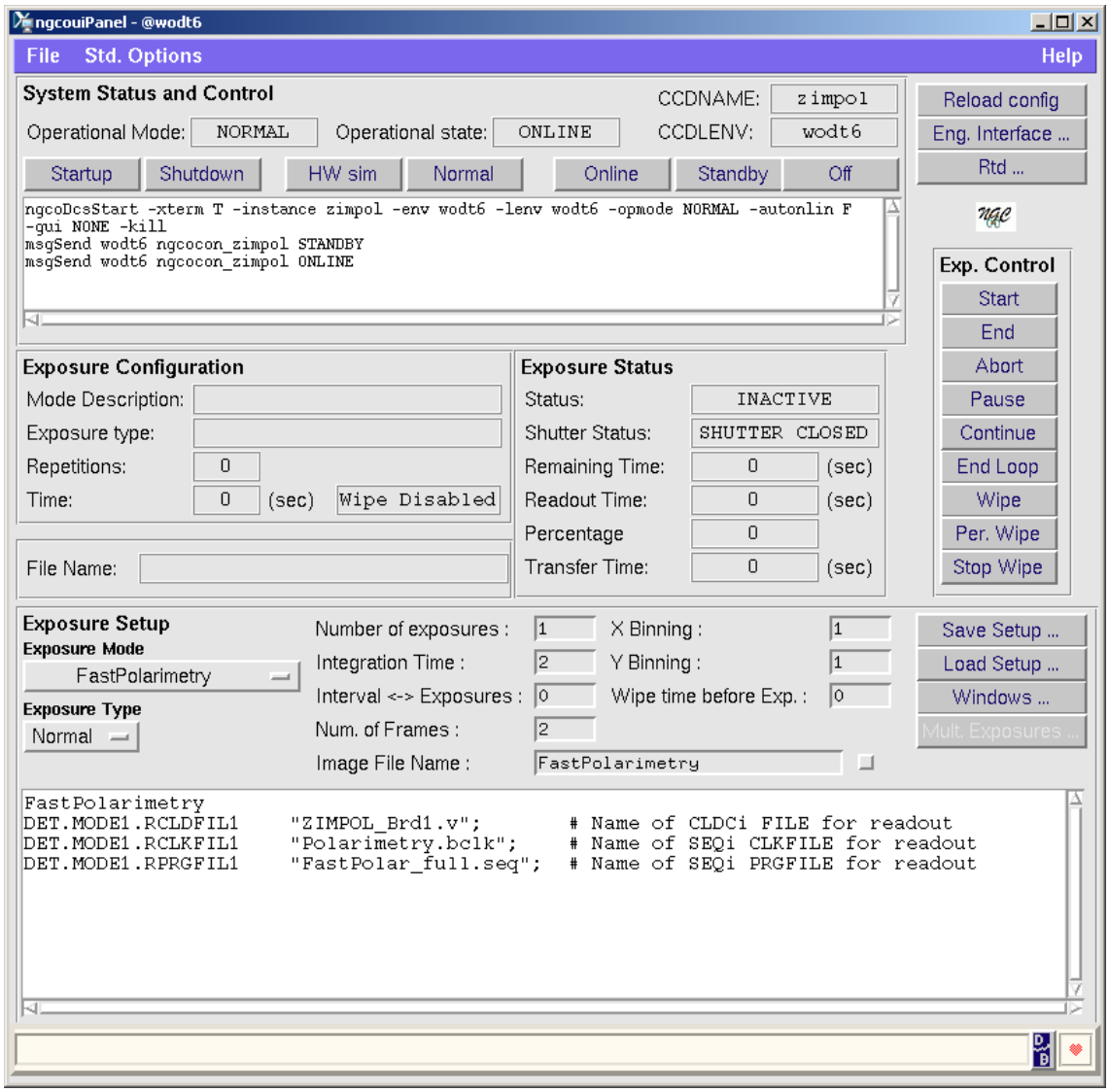


Figure 12 Optical Engineering GUI

8.4. Video Display

Figure 13 shows the video-display application used to visualize images in “real-time” (RTD - RealTimeDisplay).

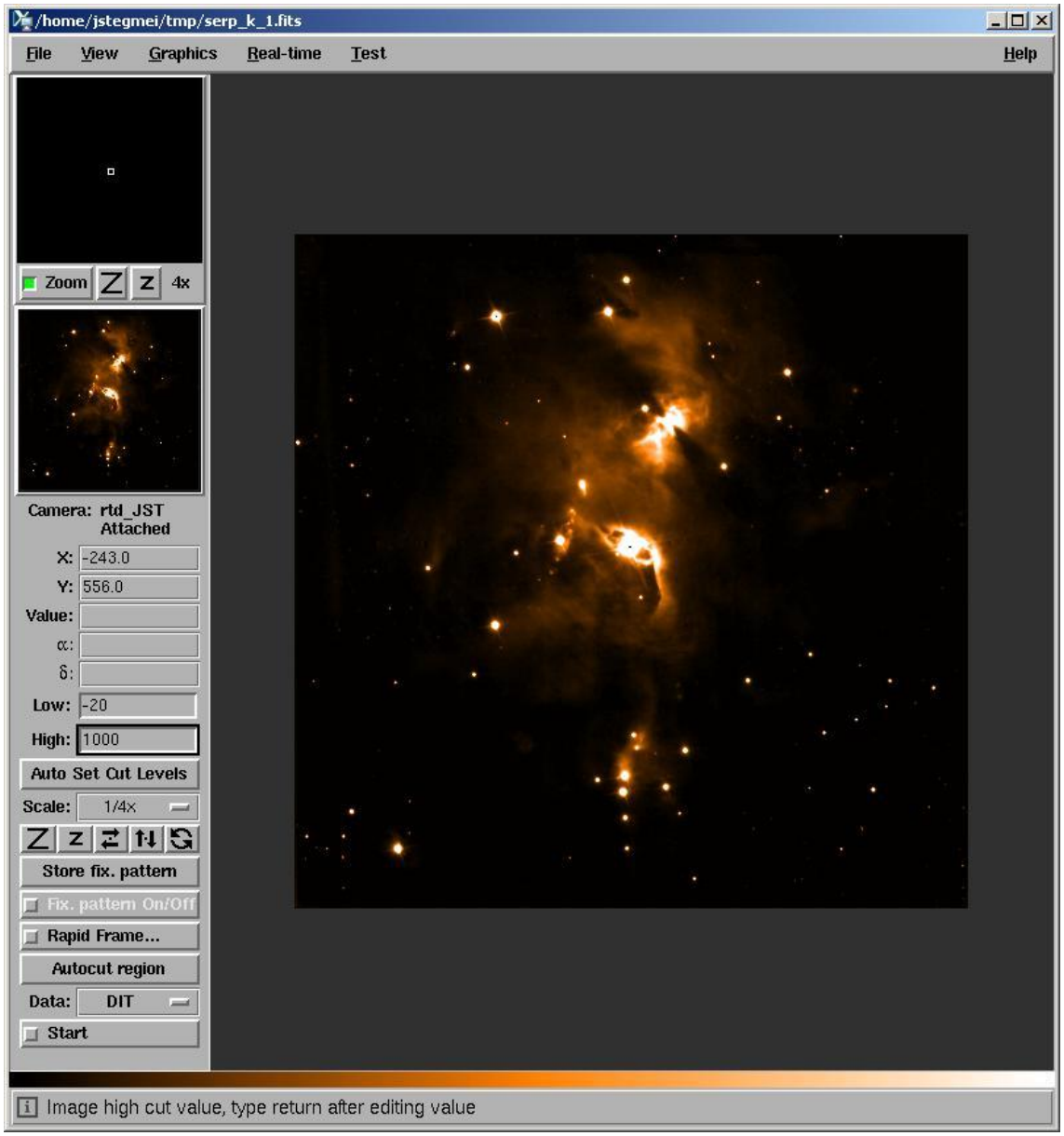


Figure 13 Video Display

8.5. Clock Pattern Editor BlueWave

For the generation of the detector clocks and sequences, a graphical editor tool has been developed at ESO: BlueWave. The picture shows a screenshot of the tool: the area on the left shows a sequence (as text), while the area on the right shows a clock pattern (as graph).

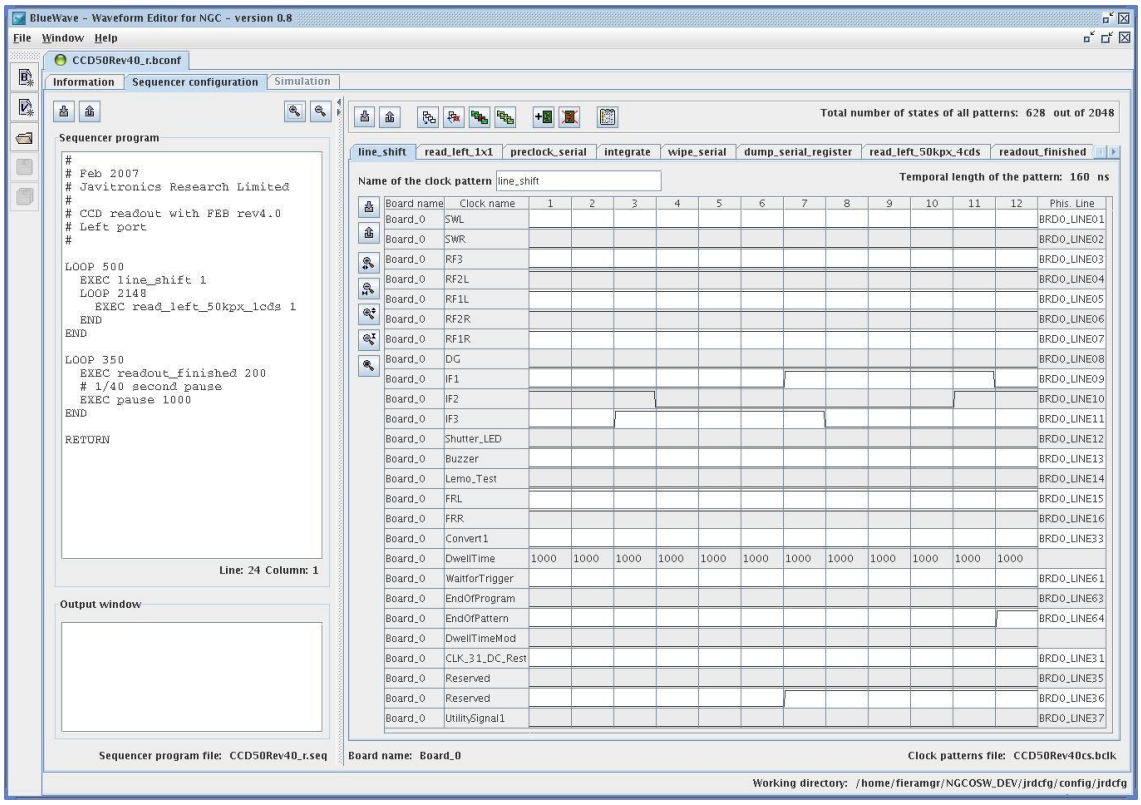


Figure 14 Clock Pattern Editor BlueWave

ACKNOWLEDGEMENTS

We would like to thank Andrea Balestra for his fundamental contribution to the development of the NGCOSW and Luigi Andolfato for his precious support and suggestions on wsf.

REFERENCES

[1] Baade, D., et al., 2009, *NGC - ESO's New General Detector Controller*, ESO, The Messenger, 136, pp. 20-24

[2] Meyer M., et al., 2009, *Detector Data Acquisition Hardware Designs and Features of NGC (New General Detector Controller)*, these proceedings

[3] Meyer, M., et al., 1996, *The ESO Infrared Detector High-Speed Array Control and Processing Electronics IRACE* , ESO, The Messenger, 86, pp.14-17

[4] Beletic, J.W., et al., 1998, *FIERA: ESO's New Generation CCD Controller*, Kluwer, ASSL, vol. 228, pp. 103-114

[5] Cumani, C., et al., 1998, *The Architecture for two Generations of ESO VLT CCD Controllers*, Kluwer, ASSL, vol. 228, pp. 115-122

[6] Wagner, F., et al. 2006, *Modeling Software with Finite State Machines. A Practical Approach*, Auerbach Publications

[7] See the UML site of the Object Management Group: www.uml.org

[8] www.sparxsystems.com.au

[9] www.magicdraw.com/

[10] Andolfato, L., *Workstation Software Framework User Manual*, VLT-MAN-ESO-17210-3872, ESO