FINUDA Note 38/LNF/DAQ/96
2 Giugno 1996

The FINUDA Data Acquisition System

P. Cerello[a], V. Filippini[b], L. Fiore[c], P. Gianotti[d], S. Marcello[a], B. Minetti[e], A. Raimondo[f] [a] INFN, Sez. di Torino, Torino, Italy.

[b] INFN, Sez. di Pavia, Pavia, Italy.
[c] INFN, Sez. di Bari, Bari, Italy.
[d] INFN, Laboratori Nazionali di Frascati, Frascati, Italy.
[e] Dip. di Fisica, Politecnico di Torino and INFN, Sez. di Torino, Torino, Italy.
[f] Dip. di Fisica Sperimentale, Università di Torino and INFN, Sez. di Torino, Torino, Italy.

Abstract

A parallel scalable Data Acquisition System, based on VME, has been developed to be used in the FINUDA experiment, scheduled to run at the $DA\Phi NE$ machine at Frascati starting from 1997. The acquisition software runs on embedded RTPC 8067 processors using the LynxOS operating system. The readout of event fragments is coordinated by a suitable trigger Supervisor. Data read by different controllers are transported via dedicated bus to a Global Event Builder running on a UNIX machine. Commands from and to VME processors are sent via socket based network protocols. The network hardware is presently ethernet, but it can easily changed to optical fiber.

1 Introduction

FINUDA [1, 2] is a non focusing magnetic spectrometer approved for construction and continuous running at $DA\Phi NE$, the Φ Factory [3] that will be operational at $INFN-LNF$ in 1997. The main goal of the experiment [1] is the fine spectroscopy of Λ- hypernuclei produced after the K^- capture at rest into nuclear targets and the study of their non-mesonic decay. The most demanding feature of the apparatus is the resolution on the charged particles momentum measurement ($\Delta p/p \leq 0.3\%$ for $p \sim 250 \div 300$ MeV/c), over a solid angle of $\sim 2\pi$ sr.

To achieve this goal a composite tracking device has been designed, constituted by one octagonal layer of silicon double-sided microstrips[4], two octagonal layers of Low Mass Drift Chambers [5] and a six layer array of Straw Tubes[6].

The full apparatus will be immersed into a He atmosphere, to minimize the multiple Coulomb scattering, and will operate inside a superconducting solenoid providing a field of 1.1 T, homogeneous within 1% over the full tracking volume.

2 General Architecture

The FINUDA Online team has developed a general purpose Data Acquisition system (FDAQ) with high flexibility and able to manage all the standard operations that a DAQ system must accomplish. The FDAQ system can be considered a multiprocessor, tree structured architecture. Data flow through a dedicated bus in the system, proceeding in the same direction. In their migration, events can be accessed for various reasons such as merging, monitoring, etc. These operations are performed by processes running in a set of distributed processors.

The FINUDA experimental setup consists of several detectors, each of them running almost independently:

- Inner/Outer Silicon Microstrip detector (SIL);
- Low Mass Drift Chambers (LMD);
- Straw Tubes array (STB);
- Internal and External Time of Flight detector (TOF);
- Global Trigger Supervisor (GTS).

The trigger is considered as a detector, although the associated electronics essentially concerns the initialization procedure and it does not contribute significantly to the event size.

Taking into account that all the frontend modules perform the zero and overflow suppresion, an event size of about 2 *kbytes* for a typical event is expected. The FDAQ is designed to reach an event rate of at least 100 Hz.

The FINUDA on-line architecture is designed to enable centralized acquisition as well as independent data taking for each detector. The modularity and complexity of the apparatus are taken into account by a distributed architecture, with a high level of parallelism, adopted in order to cope with the large amount of data.

Fig. 1 and fig. 2 respectively show the scheme of the FDAQ architecture and of the communications between all the processes involved in the system.

In order to allow fast data acquisition, microprocessors are used on the various detectors. All the signals coming from the detector's Front End Electronics (FEE) flow through a dedicated bus, using the CES VIC8250 module, and they are read in a parallel mode by one VME CPU (the local event builder) to assemble all the informations in a single sub-event and to perform a local monitoring. All the sub-events flow through a dedicated bus, using the CES VIC8251 module up to an IBM Power PC running the Global Event Builder (GEB) process.

At present it can read out all the CAMAC and VME modules used by the FINUDA detectors, but it can be easily extended to other similar situations.

The FDAQ system is a modular program consisting of some main blocks communicating with each other through socket connections:

- The Finuda Run Control (FRC), running on an IBM Power PC machine, makes use of the Tcl/Tk package [7] for the Graphic User Interface and drives all the DAQ system by sending commands, through TCP sockets, to the VME CPUs of each detector and, through an internal UNIX socket, to the Global Event Builder.

- The Global Event Builder (GEB), running on the same IBM Power PC machine used for the FRC, gets commands from it through an internal UNIX socket. Its main tasks are to read from the mirrored memories of the VIC8251 modules the pieces of events corresponding to each detector, to format the global events and to write them on tape. Periodically, the GEB receives from the Slow Control System a buffer of information concerning the general conditions of the data taking and it stores them with the physics data. In addition, it samples some events and sends them through UDP sockets to the global monitoring processes.

- The Local Event Builder (LEB), available either for FIC8234-OS9 or for RTPC8067-LynxOS CPUs, controls the read-out of VME and CAMAC modules and writes the data on the mirrored memory of a VIC8251 module. Optionally the events are also sent to a local monitoring machine through a non blocking UDP socket.

- The Global Event Monitor (GEM), running on a separate UNIX machine, gets events from the GEB and, after the reconstruction, it allows a complete overall data quality control.
- The Local Event Monitor (LEM), available either for VMS or for UNIX machines, receives some monitor events from the LEB process and performs a local, fast monitoring task, in order to check the performance stability of each detector.
- The Slow Control System (SCS), running on a separated CPU, controls and monitors the detector power supplies, gas flows, magnet performances etc.

In order to allow a dynamic change of the CAMAC and VME readout configuration, a suitable system has been developed. The list of the modules to be read is not fixed, but it can be changed before each run reading a configuration file, shared between the acquisition VME CPU and the monitoring machine through NFS. In this way there is only one source of information for acquisition and monitoring tasks, avoiding disalignments and errors. At present 8 different types of VME modules and, using a VME CAMAC branch driver CBD8210, 11 CAMAC modules can be read out. It is not difficult to add new modules thanks to the flexibility of the structure. A library of programs, written in C language, has been created in order to address VME modules using a strategy that reminds the CAMAC ESONE package.

The FDAQ system is described in detail in the following sections.

3 The FINUDA Run Control (FRC)

The FINUDA Run Control process (FRC) allows the operator to drive the whole data acquisition system, taking advantage of a Graphic User Interface based on the Tcl/Tk package.

The possible stable states of the system and the commands connecting them are shown in fig. 3. A Tcl/Tk window corresponding to each stable state (SLEEP, READY, ACTIVE) is available to the operator for sending commands to the detectors and the GEB. The main window of the system has a three option structure: the green button starts the operations by trying to open all the sockets defined in the program, the yellow button gives access to the tape management, while the red one stops the program execution. When the green button is chosen, an intermediate window tells the operator whether the socket connections were successful or not, allowing to proceed or to try again. In the first case, the SLEEP state is accessed and the corresponding window is displayed (fig. 4). In this state, the main run conditions are chosen: the required detectors are selected and the triggers are chosen, each one with an optional corresponding prescaling factor. This window gives access to two possible commands: SETUP and STOP Run Control. When all the parameters have been set, the operator can send the SETUP command to the selected detectors: the FRC waits for the answers and, when all of them have been correctly received, deletes the SLEEP window and prompts the READY window (fig. 5). This state gives access to the START and the RESET commands, and requires the choice of the output device (disk, tape or dummy): in case of missing choice, the program does not accept a START command. This command is sent to the Global Event Builder (GEB), the Global Trigger Supervisor (GTS) and the detectors in three following steps, each starting only after a positive answer from the previous one. In this way, the arrival of triggers before the detectors are ready to take data and the collection of events before the GEB is ready to manage them are avoided. When all the detectors send back a positive answer to the FRC, the READY window is deleted and the ACTIVE and MONITOR windows (fig. 6) are displayed. The ACTIVE window monitors some of the basic run variables (the number of collected events, the event length, the amount of data written on tape, etc.) and it is updated every second. It is possible to check new parameters with simple modifications of the code. The MONITOR window gives access, on request of the operator, to some very significant histograms: the buffer length distribution of each detector and the global one, the event rate and the data flow, sampled every 5 seconds, as a function of time during the last 10 minutes.

The STOP command is the only operation allowed in this state: it distributes the command to all the detectors and gives back the READY window. At this stage, it is possible to directly start a new run or to send a RESET command to the detectors, if some basic modifications of the run configuration are needed.

4 The Global Event Builder (GEB)

The main task of the Global Event Builder process is to collect the event pieces coming from different detectors and to build up the complete event to be written on the output device and to be sent to the GEM. Moreover, the GEB periodically receives a buffer from the Slow Control System and adds it to the information to be written on tape. It is running on the IBM Power PC CPU and it is connected to the FRC process, in order to receive commands, through an internal UNIX socket; to the Global Event Monitor (GEM) and to the Slow Control System (SCS) through UDP sockets.

UDP sockets don't need an effective connection between the CPUs involved in the communication: in this way the GEB program can run even if no monitoring processes are active. On an UDP socket the packets are sent without verifying that on the line there is an active process receiving them. This is useful in order to speed up the GEB program and to make possible the parallel run of different monitor programs.

The GEB starts with an initialization phase, in which the VIC8251 mirrored memories are cleared and initialized and the output device is defined; then it waits for the START command coming from the FRC. Once the data taking has started it looks at the VIC8251 mirrored memories in order to check whether the event pieces are ready; it reads them and, after having checked their alignment, it formats the global event and sends it to the writing procedure, which takes advantage of the Asynchronous Input Output (AIO) facility, provided by the operating system.

The VIC8251 mirrored memories, corresponding to each detector, are organized into 8 buffer levels: that gives a pipeline structure to the data acquisition system and allows to minimize the dead time.

The GEB process makes possible the monitoring of the most important variables and distributions related to the data acquisition performance through a memory shared with the FRC. There it periodically writes the information which is then read by a specific Tcl/Tk function called by the operator from FRC.

5 The Local Event Builder (LEB)

The main task of the Local Event Builder process is to collect data from the front end electronics of each detectors and to make the information available to the GEB by writing on the 4Mb mirrored memory of a VIC8251 module. The LEB processes corresponding to each detector run on separate VME CPUs, connected with the FRC through TCP sockets. When the LEB receives a command it performs some operations and, if all of them are successfully completed, an acknowledge message is sent back to the FRC. This message is the identifier of the stable state to which the LEB program has gone.

The SLEEP state is the LEB starting point. When the LEB is in that state the run has not been configured yet; the READY state occurs after a SETUP command, as soon as all the front-end modules have been initialized and prepared for the read-out. When the LEB receives a START command it goes to the ACTIVE state, where the data read-out and transfer goes on.

If something goes wrong, the state of the process becomes FAILURE. The only exception to this scheme is the START command which sends back to the FRC not only the final state, but also a buffer of data containing all the information concerning the front-end initialization (discriminator thresholds, TDCs status word etc...).

This buffer is structured with an header part, containing general information as buffer length, run number, event number, detector identifier, etc., followed by the data, structured into equipments, one for each module type.

5.1 FEE Initialization

The module list has been created to make possible a simple run configuration by editing a data cards file. In this way it is not necessary to recompile the DAQ program if the list of modules to be read has been changed. The file is written in free format.

To exclude a module from the read-out it is enough to give it a null base address (if it is a VME module), or a null crate number (if it is a CAMAC module).

Checks of consistency are made on the read data in order to verify the goodness of the information written in the module list.

The electronics initialization is done separately for VME and CAMAC modules by two similar routines. They make a simple reset of the modules or, where it is needed, they configure the modules following the indications taken from the module list file. Moreover, hardware settings are compared with those written in the module list (i.e. VME scaler cascading) in order to check if the run has been properly configured.

5.2 Event read-out

The read-out procedure is organized in order to construct the data buffer in parallel with the CAMAC and VME modules read-out. This buffer is then written in the mirrored memory of a VIC8251 from which the GEB program will take it. After a START command has been received from the FRC program, a polling is executed to check the arrival of a trigger signal on the CORBO module or the arrival of a STOP command from the FRC. The trigger signal starts the read-out, while a STOP command brings back the program to the READY state.

The event buffer is structured as previously described.

6 VME Libraries

In order to access VME modules in a simple and transparent way a library of programs, written in C language, has been developed. The modules that are presently recognized by the library are:

- Corbo;
- Multi-Hit Pattern Unit;
- 16 Channel Scaler (32 bits);

- 8 Channel Gate & Delay Generator;
- Dual Programmable Logic Unit (PLU);
- 16 Channel I/O Register;
- 64 Channel I/O Multi QDC;
- 96 channels TDC;
- 2 channel C-RAMS FADC.

Each library function takes care of controlling the validity range of the parameters used and gives back an error message, if the action required is not possible, or a warning message, if this operation is dangerous.

Concerning VIC8251 a library has been created to allow the transfer of the event fragments to the GEB. There are three main functions in the library. The first one initializes the VIC8251 and prepares a block pointer: it is called only once at the beginning of the run. The second one returns a block pointer to the mirrored memory in which the LEB could write its part of event. Finally, the last one closes the event writing its length at the top of the buffer.

The program uses also the following standard libraries provided by the CES company:

- CES VIC8250 - VMV to VME One Slot Interface
- CES CBD8210 - Branch Driver

7 Conclusions and Performances

A DAQ system prototype was assembled and it is under test. The acquisition rate with a simulated buffer of 2 $kbytes$ reaches 100 Hz.

The described FDAQ system is based on commercial hardware standards like CAMAC and VME. The consistent use of the standard VIC bus as data path, between FEE and EBs, has led to a high degree of flexibility in the DAQ architecture. The use of an external module list, read as data card, allows the change of the hardware configuration avoiding a new compilation of the program. Furthermore the modular run control software, with its high level man/machine interface, provided by the Tcl/Tk package, makes the overall DAQ extremely user friendly.

Thanks to its open structure the system can grow with new demands and can be adapted to new technologies.

References

[1] The FINUDA Collaboration, INFN Report LNF-93/021 (1993).

[2] The FINUDA Collaboration, Technical REport, INFN Report LNF-95/024 (1995).

[3] L. Maiani et al *The second* DAΦNE *HandBook* **I-II**, LNF-INFN (1995)

[4] L. Celano *et al*, CERN-PPE/95-106, accepted by *Nucl. Instr. Meth.*

[5] M. Agnello *et al*, *Nucl. Instr. Meth*, **A367**(1995)100.

[6] L. Benussi *et al*, *Nucl. Instr. Meth*, **A361**(1995)180.

[7] John K. Osterhout, *Tcl and the Tk Toolkit*, Addison-Wesley 1994.

List of Figures

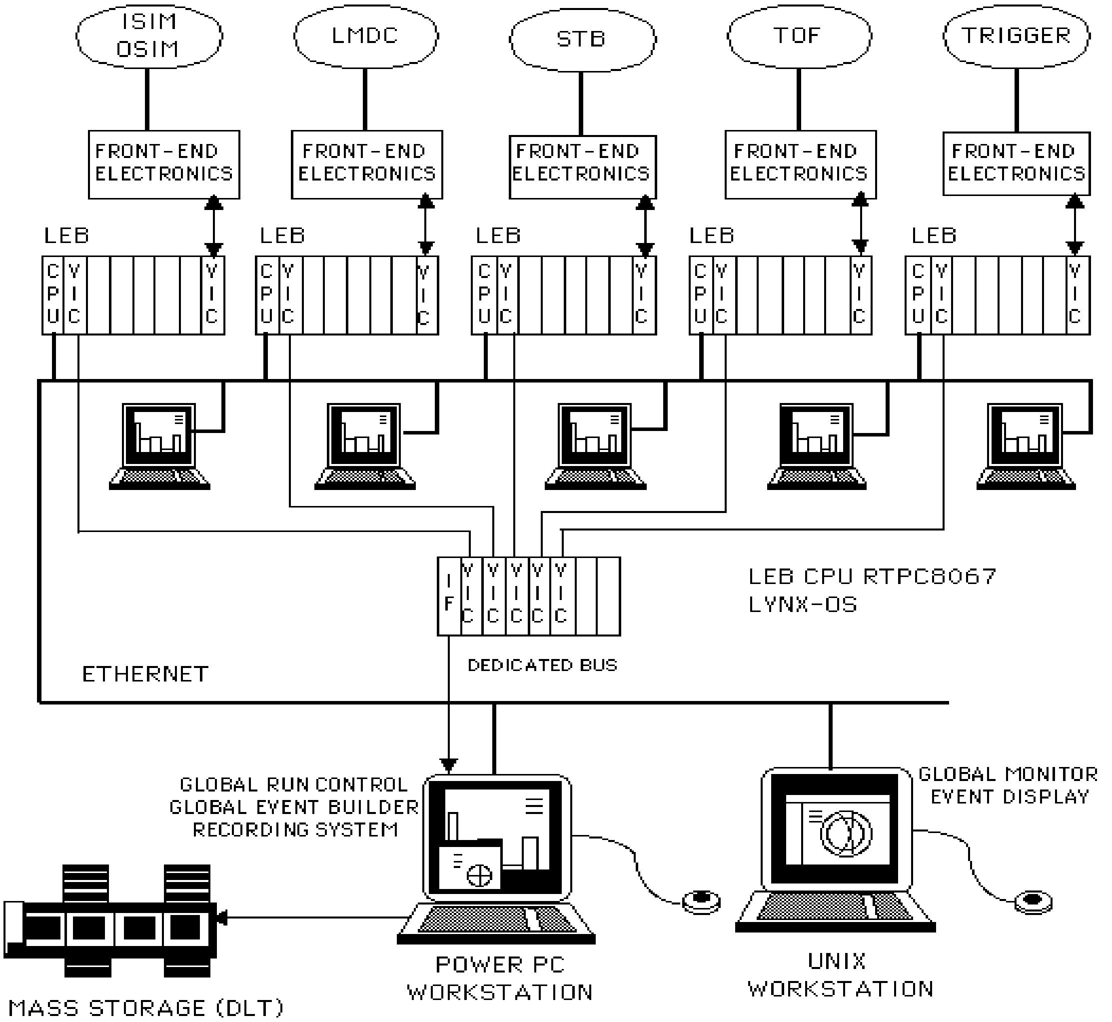

Figure 1: The FDAQ hardware architecture.

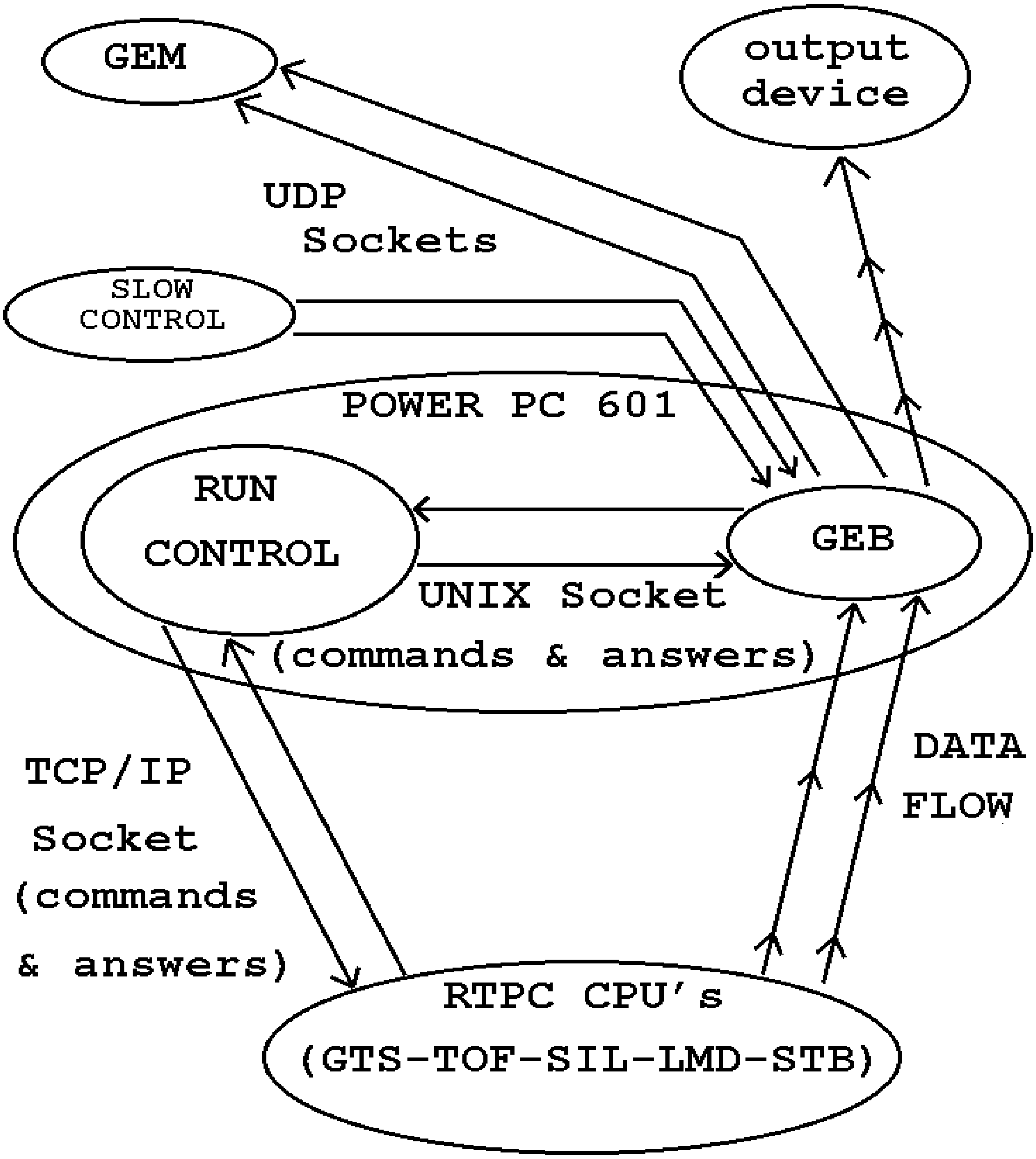

Figure 2: The processes involved in the FINUDA data acquisition system and their communication scheme.

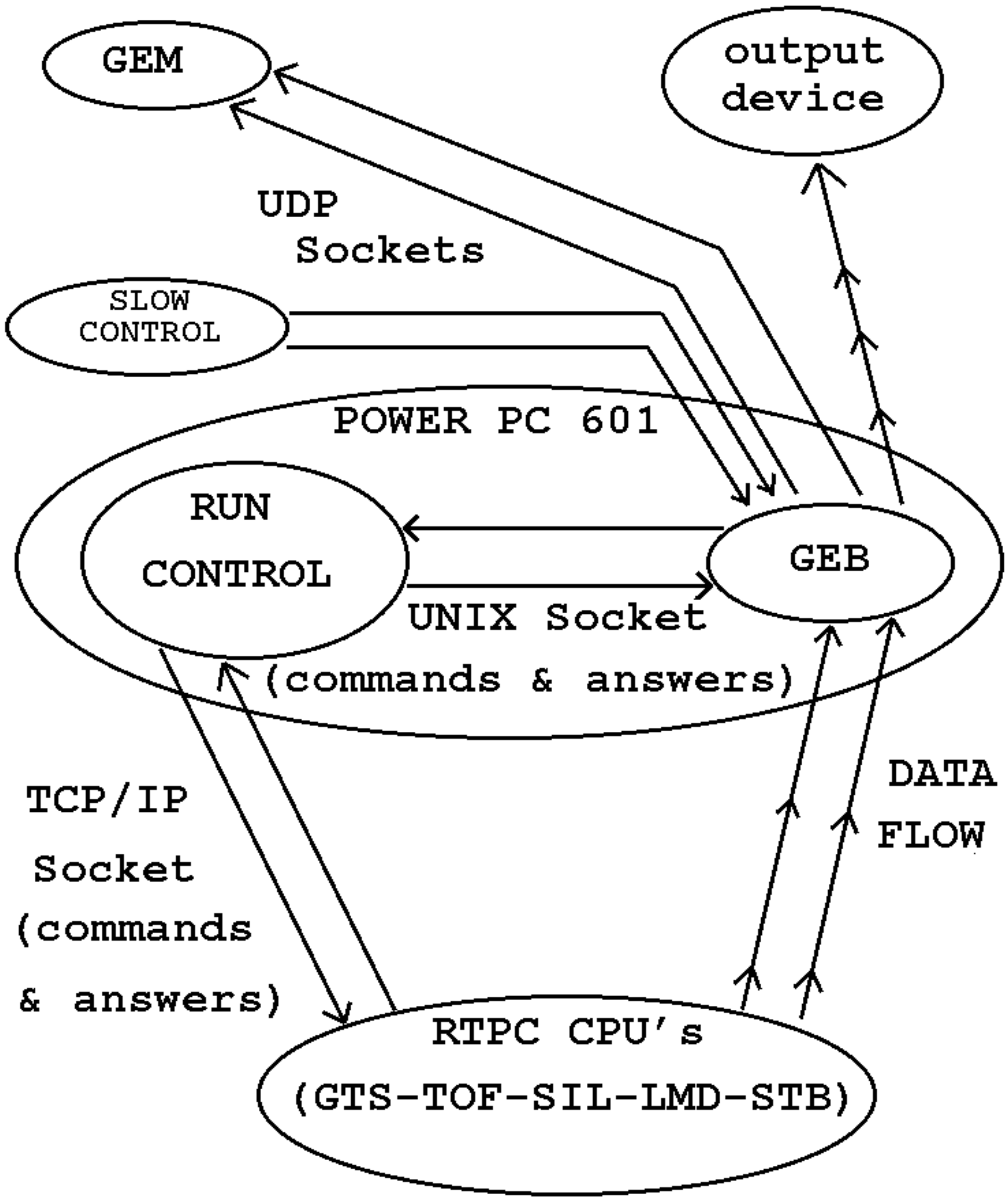

Figure 3: The possible states of the FINUDA Run Control and the commands connecting them.

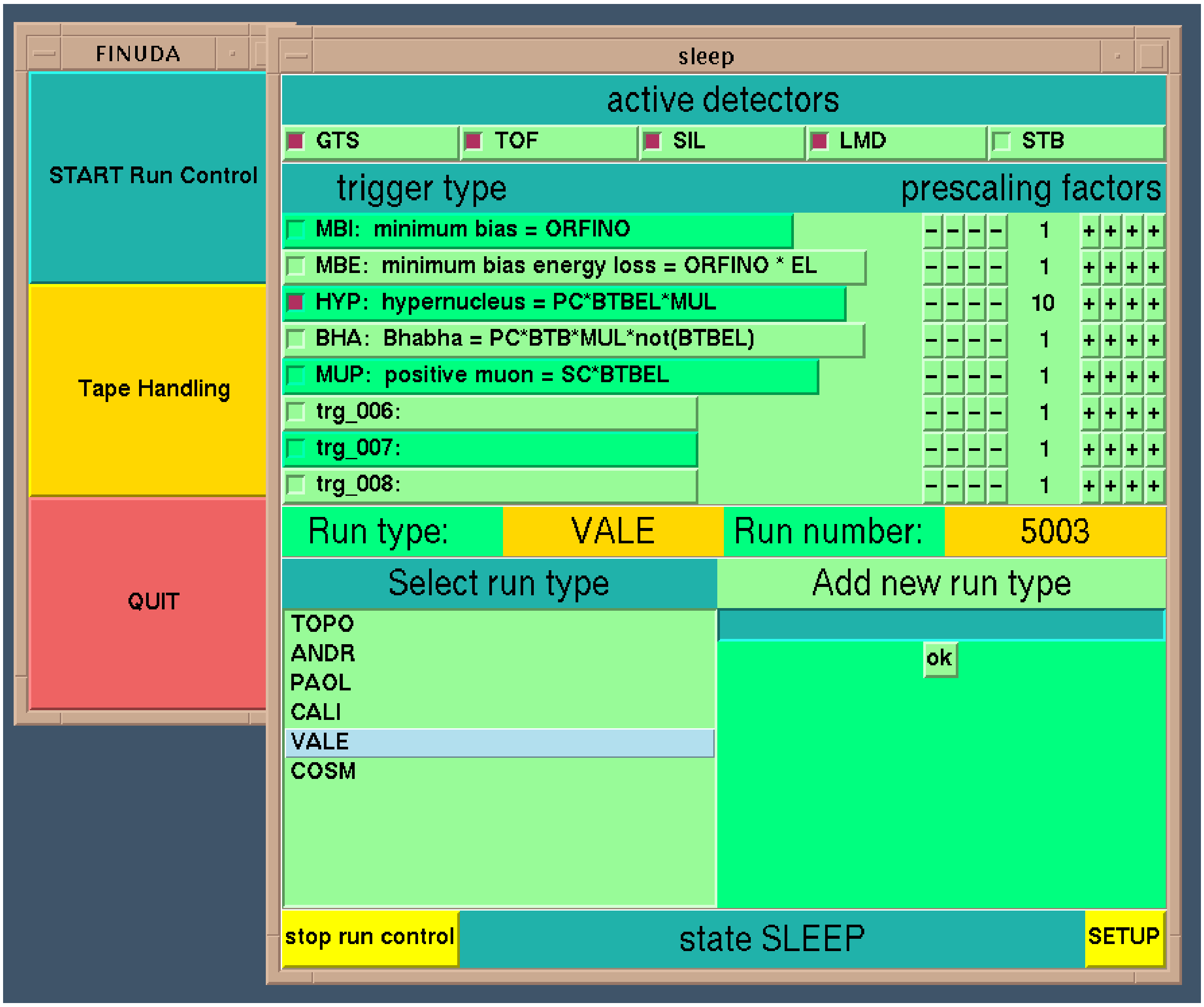

Figure 4: The SLEEP window.

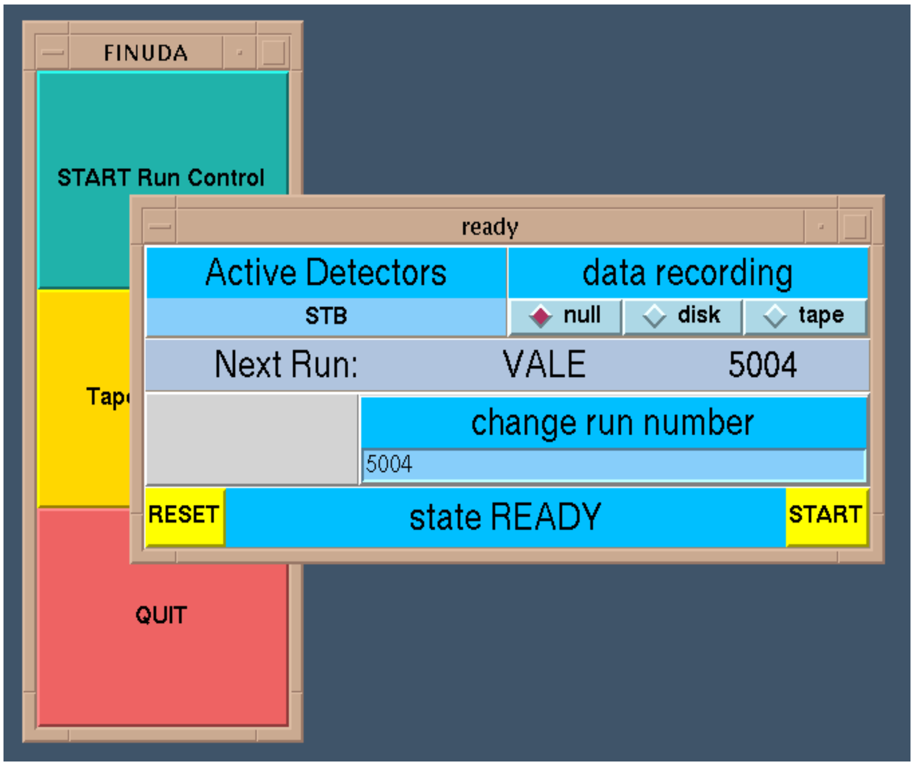

Figure 5: The READY window.

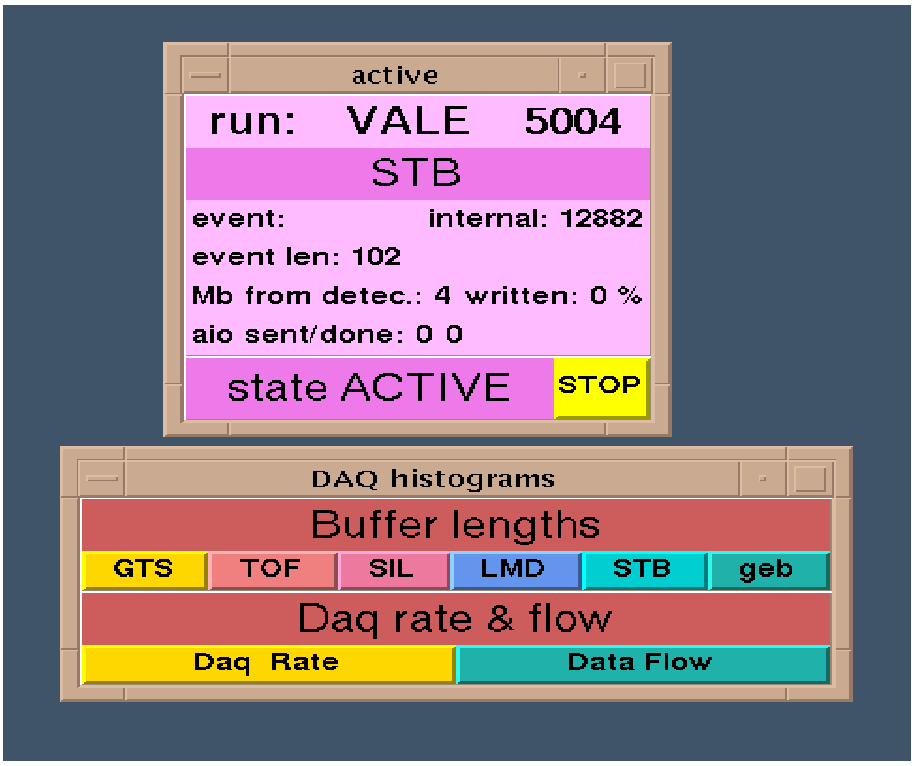

Figure 6: The ACTIVE and MONITOR windows.