daq Working Group
Tech. Note: 01 Rev. 0
March 1998
Supersedes:
Superseded by:

A. D'Ambrosio
S. d'Angelo

GNOsys
The data acquisition system for the G. N. Observatory

Status of this memo:

This document is intended for a general audience; it describes the guideline followed in to design the GNO's data acquisition system and it enters in some details of the actual implementation. In this it reflects the status as at the end of January 1998.

Università degli studi di Roma "Tor Vergata"
Dipartimento di Fisica
rom2f/98/07 (feb.98)

Contents:

general description

The main goal is to provide the G. N. Observatory with a data acquisition/distribution system that is speedy, reliable and highly available. In addition the system should stay updated for years and the upgrades should be backward compatibles as much as possible.

To guarantee reliability and *forward* compatibility we adopt an extremely modular design, based on well established standards. To assure availability we duplicate every critical component.

The whole system is, logically, made of three parts: an *on-line* subsystem that directly dialogues with the electronics, controls the instruments and store data on disks; an *off-line* subsystem that performs some homekeeping and distributes data to remote clients; a *mass storage* subsystem on which all data resides. The fig. 1 shows the schema of the planned hardware.

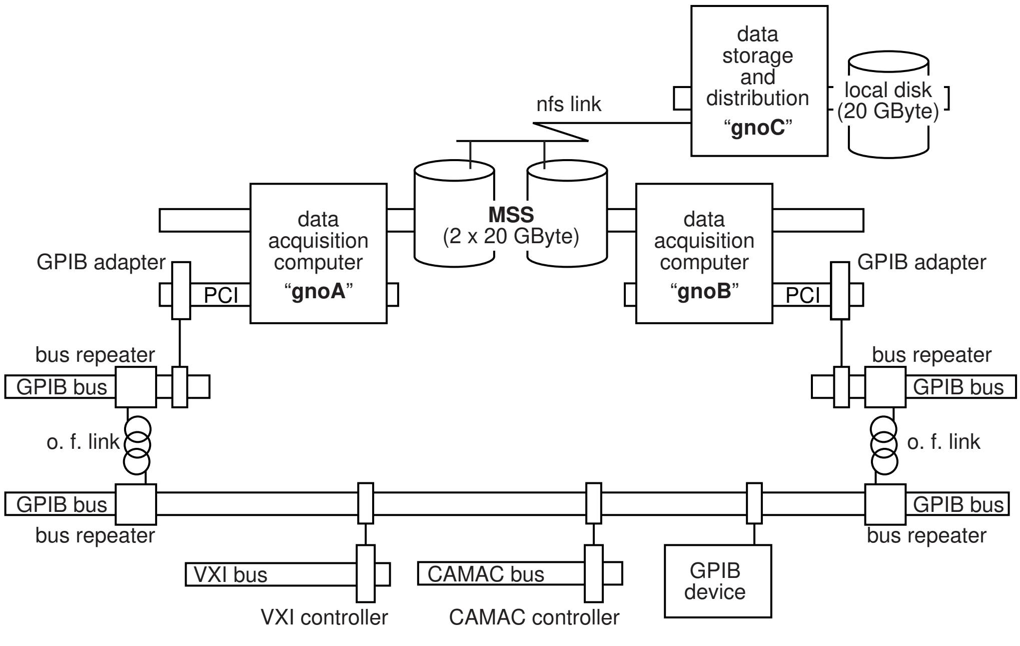

Fig. 1 — Schema of the data acquisition system

One UNIX machine (`gnoC`), equiped with 20 GByte local disk, realizes the off-line subsystem; the mass storage `MSS` offers 40 GByte and it is built with twelve 4.3 Gbyte disks arranged in two RAID-V sets. It is locally mounted in the on-line and it is remotely mounted (via `nfs`) in the off-line.

The on-line system is composed of two identical UNIX boxes (`gnoA` and `gnoB`) that share the Wide-SCSI bus on which the disks reside. Each machine control the same IEEE 488.2 bus through a PCI/GPIB adapter. A pair of repeaters, linked by an optical fibre cable, is used to extend the bus to more than 15 meters (up to 1 Km) and to electrically insulate the second trunk. It also increases the number of GPIB addresses from 16 to 32. Two addresses being reserved to the adapters on the `gnoA` and `gnoB` machines; the bus can host up to 30 IEEE 488/488.2 compliant devices. Even if there aren't GPIB adapters for VXI or CAMAC buses, single VXI or CAMAC crates can be driven using GPIB/VXI and GPIB/CAMAC crate controllers.

The components of the on-line software and their relationship with the operating system and with the electronics are schematically shown in fig. 2. A *device driver* is installed into the OS kernel communicates with the PCI/GPIB adapter allowing the *daq daemon* (`daqd`) to exchange messages (commands and data) with the GPIB bus.

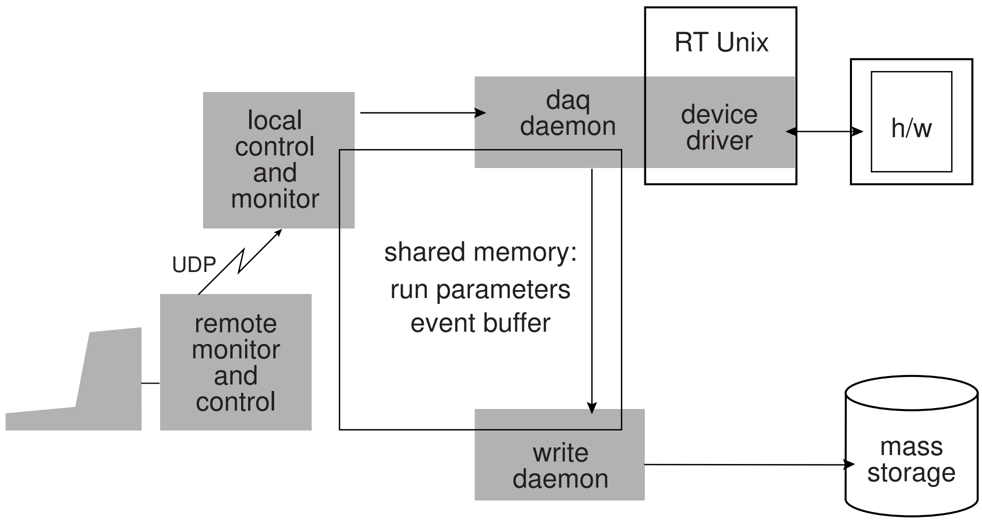

Fig. 2 — Schema of the software components

The `daqd` reads commands from and stores data into a non volatile *shared data area* (`sda`) that is in common with the *write daemon* (`daqwrt`) and with the *local controll* process (`daqctl`). While `daqwrt` performs an unique task: to flush on disk all available data at every signal received from `daqd`; the tasks of the local control are: i) to issue signals for to start/stop both the demons; ii) to fill the `sda` with the actual values of the parameters (run number, run type, ...) to be used in the run; iii) to send relevant information about the status of the system to the *remote control* process. This only is in charge of the man-computer dialogue: presentation of the information on screen and interpretation of key-strokes for to steer the local control.

hardware

Table 1 lists the quantities needed (Qty) and presently available (Avl) for any relevant component of the system.

Table 1 — Part list

Qty	Avl	Description	
1	0	α-server 2100	(D:)
		includes PCI/SCSI adapter	(D:)
		includes Ethernet I/F	(D:)
1	1	+ 128 Mbyte memory expansion	
1	0	+ PCI/SCSI adapter	(D:KZPSC-BA)
1	1	powered cabinet	(D:H9A10 metric)
2	1	α-server 1000	(D:ASR1000A 5/400)
		includes PCI/SCSI adapter	(D:)
		includes 2 Gbyte disk	(D:)
		includes Ethernet I/F	(D:)
2	1	+ 128 Mbyte memory expansion	
1	1	+ DAT tape	(D:TLZ90)
2	1	+ PCI/SCSI adapter	(D:KZPSA-BB)
2	1	+ PCI/GPIB adapter	(NI:777260-01)
1	1	controller shelf	(D:BA350-MA)
2	1	SCSI controller	(D:HSZ50)
2	1	+ 32 Mbyte cache memory	
2	1	+ cache memory backup battery	
2	1	16-Bit SBB shelf	(D:BA356-S)
2	1	16-Bit I/O module	(D:)
20	6	4.3 Gbyte disk Wide-SCSI	(D:DS-RZ1CB-VW)
4	0	GPIB extender	(NI:GPIB-140)
4	0	50 mt optical fibre cable	(NI:182805-010)

The term in parenthesis in the last column indicates the supplier (D: for Digital, NI: for National Instruments) and the supplier's item number.

At the time of this writing:

`MSS` — 20 Gbytes are assembled in the GALLEX laboratory in Rome. The order for the rest was already placed by the Milano group.

`gnoC` — the machine is in the ffl of the GALLEX building; the OS must be changed to UNIX (presently is VMS); the disks must be upgraded and another SCSI controller must be added. No actions can be taken while it is in use for the GALLEX data analisys.

`gnoB` — the order for this machine was already placed by the Milano group.

`gnoA` — the machine is assembled and is functioning in the GALLEX laboratory in Rome.

assembling and cabling

Assembling and cabling `gnoA` is in some extension straightforward; most of connections being via internal buses. In fig. 3 it is shown the front view of the cabinet; the white rectangles indicate the position of the installed items, the shadowed ones indicate the place-holders for the next devices to install. Two and half units are still free for future expansions (if needed).

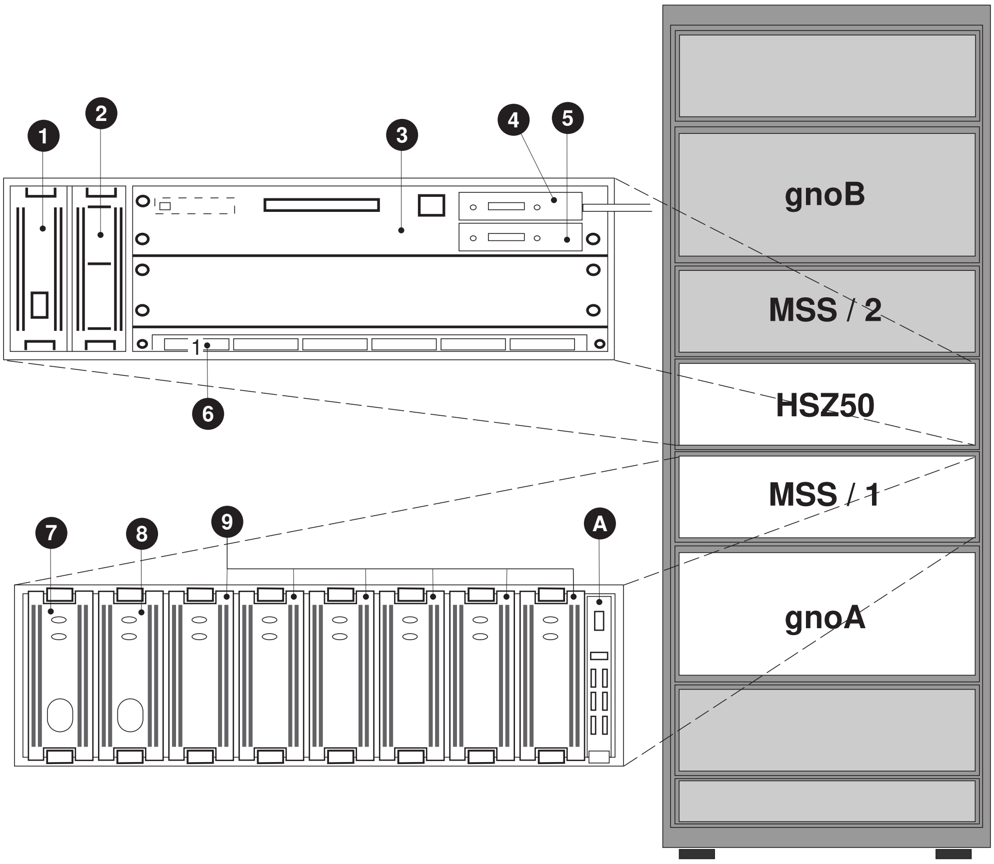

Fig. 3 — Front view of the cabinet

The expanded views (at left in fig. 3) show the position of modules inside the shelves, most of positions being obliged. Item (1) is the power supply and must occupy the leftmost slot; item (2) is the cache memory battery backup, it could be anywhere (and in fact — in this position — it steals the place of the power supply backup). The HSZ50 in the upper position (3) assumes the SCSI address of 7 (the second one will be put, in the same shelf, in the lower position at SCSI address of 6 and this will prevent the slot (8) from to be used for a hot spare disk). Cable (4) comes from the KZPSA-BB controller and the bus is (presently) ended by the terminator (5). The position in the shelf of the six disks (9) fix their SCSI address (0 is the rightmost one, 5 the leftmost). Finally one cable connect the controller port 1 (6) with the i/o-module (A).

devices configuring

The six disks — (9) in fig. 3 — are automatically configured and identified by the HSZ50 controller as: DISK100 .... DISK150 from right to left (their ids are needed to be known in case of manual replacement). A single RAIDSET (RAID1) is made at the PTL of 100; this and the number of the SCSI bus uniquely defines how the OS will name (`rzb8`) the device.

Table 2 — Device assignements on `gnoA`

`/dev/...`	is on:	
`ln0`	eisa 0	ethernet i/f
`rz0`	scsi 0 target 0 lun 0	2 Gbyte system disk
`rz4`	scsi 0 target 4 lun 0	cd-rom
`tz5`	scsi 0 target 5 lun 0	DAT tape
`rzb8`	scsi 1 target 0 lun 1	20 Gbyte RAID-V disk
`ib0`	pci 2 slot 3	PCI/GPIB adapter

Table 2 lists the assignements for each relevant device recognized on `gnoA`; it may be convenient, even if not mandatory, to maintain same assignements on the other machines.

Table 3 — Disk partitions

	offset	size	type	cylinders
`rz0a`	0	262 144	4.2BSD	0 – 63
`rz0b`	262 144	1 048 576	swap	64 – 319
`rz0g`	1 310 720	7 067 648	4.2BSD	320 – 2 045
`rzb8c`	0	41 879 900	4.2BSD	0 – 18 530

As shown in Table 3, the system disk has been partioned in non standard way, in order to increase the swap space and to regain space from the h-partition; the raid disk is instead used as only one “big” partition.

Achille’s heel

The system looks very robust, however there are few weak points. As matter of fact it can be blocked by:
– a failure in the UPS (to prevent this it would be necessary only a second UPS; in fact redundant power distribution is already foreseen).
– the fault of two disks in the same shelf at same time (to prevent this it would be necessary a third shelf and at least one more disk).
– the fault of one i/o-module (to prevent this it would be necessary to have four more shelves with their i/o-modules).

Ça va sans dire that fault tolerant systems does not guard against accidental deletion of files (in UNIX-like OS a deleted file cannot be rescued; it is simply gone). This means that periodic backups are always needed.

Finally, even if there are two of each composant it is not a good habit to replace a faulty module on a system with the working one in the other (often this operation would damage the second one too); the right step is to swap from one system to the other. Consequently the software on both machines sould be the same at any time.

software

system software

From here all considerations refer to `gnoA` machine only. Presently it runs Digital Unix (formerly known as OSF/1) operating system, version 4.0B (Rev. 564). The kernel was build accepting only the following options:
– Sistem V support
– Kernel Brkpt
– Packet filter driver
– STREAMS
– X/Open transport
– ISO 9660 CDFS
– Audit.

the resulting configuration file is: `/sys/config/GNOA`.

Some resource consuming products (eg. Netscape navigator) that come with the base software was not explicitly removed but certainly will not be upgraded and their use on this machine is strongly discouraged.

system software additions

In addition to the base system software, the following products have been installed:

i) using the *setld* command (from the distribution media):
– dfa500 (Digital: Fortran 77 compiler. Vers 5.0.0)
– lps510 (Digital: laser printer server. Vers 5.1.0)
– swa11a (Digital: remote consolle for HSZ50. Vers 1.1A)
– PCIGPIB (National Instruments: PCI/GPIB driver. Vers. 1.1)

ii) sourcing from `ftp://cis.uniRoma2.it/unix/misc/dUnix_bin`
– `/usr/local/bin/bash` (the *bash* shell)
– `/usr/local/bin/perl` (the *perl* programme)
– `/usr/local/bin/sedt` (a screen editor, similar to EDT)
– `/usr/local/bin/ncftp` (a screen oriented ftp client)
– `/usr/local/bin/gnuplot` (the gnu (FSF) plotting programme)
– `/usr/local/bin/kermit` (the *kermit* data transfer programme)
– `/usr/local/bin/xa2ps` (a filter for to print *text* files)
– `/usr/bin/X11/xperfmon+` (a system performance monitor)

system software customizations

Few customizations have been made to the system software: some of these should be reconfigured when moving the system at the Gran Sasso Laboratory.

– The second subfield of the *gecos* field of user records in the file `/etc/passwd` has now the meaning of a three-letter nickname allowing the user to control the start/stop of the runs.

– Two mount points, `/d1` and `/d2` have been created. The file `/etc/fstab` has been modified for to mount the device `/dev/rzb8c` on `/d1`.

– The file `/etc/syslog` has been modified to allow the data acquisition task to log messages.

– The `lp` queue should be re-defined to point to a convenient print server.

– The internet machine names (the domain name) and addresses should be redefined as:

`gnoA.lngs.infn.it`	`192.84.135.148`
`gnoB.lngs.infn.it`	`192.84.135.149`
`gnoC.lngs.infn.it`	`192.84.135.56`

the machine `gsnet0.lngs.infn.it` (`192.84.135.16`) will be the primary nameserver while `cscgs0.lngs.infn.it` (`192.84.135.7`) will provide the default gateway.

daq software

The directory tree structure of the data acquisition system is shown below:

`/GNOsys/bin/`	all executable programmes and scripts;
`/GNOsys/lib/`	all object (`.o`) files and shareable libraries (`.so`);
`/GNOsys/src/`	all programme's sources (`.c`);
`/GNOsys/src/inc/`	all *include* files (`.h`);
`/GNOsys/var/`	shared memory and other miscellaneous;
`/GNOsys/var/logs/`	log files from all processes;
`/GNOsys/var/pids/`	lock files (the programme's *pid*;
`/GNOsys/rawdata/`	raw data (what else?);
`/GNOsys/users/`	users' login area;

The root is physically mounted at `/d1/` but is referenced only throughout the symbolic link `/GNO`; while the users' login area is referenced throughout the symbolic link `/GNOusr`.
There are *Makefile* files in the directories `/GNO/bin`, `/GNO/lib` and `/GNO/src`; to recompile and to relink tasks it is sufficient to issue the command: `make` from the `/GNO/src` directory.

the shared memory

The nucleus of the data acquisition is in the *shared data area*. It consists of $(2 + 8 \times 8)$ memory pages (one page is 8 Kbyte); the first two contain the relevant parameters for the current processes and the current run. Starting on the third page (page two) there are eight blocks of eight pages each; these blocks are linked to form a circular list of event buffers. The details of the structure are shown in the fragment of code in table 4, adapted from the comments in the include file: `inc/daqconf.h`.

Table 4 — Shared data area structure

```
/* SDA.                              */  <- process parameters
/* creation time                     */
/* creator                           */
/* privileged ctl pid                */
/* active daemon pid                 */
/* active writer pid                 */

/* RUN.                              */  <- run parameters
/* run number            [total]     */                          (a)
/* run type                          */                          (b)
/* run number            [type]      */                          (c)

/* current run start oper            */
/* current run start time            */

/* mask of required devices          */
/* line definitions                  */     x 24 |
    /* input is connected ? (T/F)    */          |
    /* counter number                */          |
    /*        type                   */          |
    /*        gas                    */          |
    /*        pressure               */          |
    /*        copper box             */          |
    /*        lead shield            */          |
    /*        pos. in tank           */          |
    /*        HV (helipot)           */          |
    /*        HV                     */          |
    /*        main amplifier         */          |
    /*        shape amplifier        */          |
    /* description                   */ ------+                  (d)

/* current run end oper              */                          (e)
/* current run end time              */                          (f)

/* write data on disk                */
/* output filename                   */

/* placeholders for next run         */
/*       editable parameters         */
```

```
/*              ditto           */
/*              ditto           */

/* "official" trigger time      */
/* "official" trigger number    */

/* 2nd last event time [total]  */
/* last event time     [total]  */

/* event number        [line]   */
/* 2nd last event time [line]   */
/* last event time     [line]   */

/* number of "red" buffers      */
/* number of "green" buffers    */
/* "purple" buffer (in use)     */
/* ptr to next "red" buffer     */
/* ptr to next "green" buffer   */

/* BUF0                         */  <- buf 0  <<---+
/* next buffer address          */  >>----------+  |
/* size of event data           */              |  |       (g)
/* event number                 */              |  |
/* event time stamp             */              |  |
/* raw data begins here: "D1"   */              |  |
/* byte count for device 1      */              |  |
/*      "E1"                    *               |  |
        byte count for eqpmt 1  *               |  |
        ...                     *               |  |
        eqpmt 1 data            *               |  |
        "E2"                    *               |  |
        byte count for eqpmt 2  *               |  |
        ...                     *               |  |
   . . .                        *               |  |
   "D2"                         *               |  |
                . . .           */              |  |       (h)
                                                |  |
/* BUF1                         */  <- buf 1 <<-+  |
/* next buffer address          */  >>----------+  |
/* size of event data           */              |  |       (g)
/* event number                 */              |  |
```

```
/* event time stamp            */             |  |
/* raw data begins here: "D1"  */             |  |
/* byte count for device 1     */             V  |
/*      "E1"                   *              2  |
        byte count for eqpmt 1 *                 |
        ...                    *                 |
        eqpmt 1 data           *                 |
        "E2"                   *                 |
        byte count for eqpmt 2 *                 |
        ...                    *                 |
   . . .                       *                 |
   "D2"                        *                 |
             . . .             */                |
                                                 |
      ... ... ...                                |       (h)
                                              |  |
/* BUF7                        */  <- buf 7 <<-+  |
/* next buffer address         */  >>-------------+
/* size of event data          */                        (g)
/* event number                */
/* event time stamp            */
/* raw data begins here: "D1"  */
/* byte count for device 1     */
/*      "E1"                   *
        byte count for eqpmt 1 *
        ...                    *
        eqpmt 1 data           *
        "E2"                   *
        byte count for eqpmt 2 *
        ...                    *
   . . .                       *
   "D2"                        *
             . . .             */                        (h)
```

the read and write daemons

`/GNO/bin/daqd` and `/GNO/bin/daqwrt` form a producer-consumer pair. The first fills an event buffer *reading* data from the GPIB bus; the second empties the buffer *writing* data on disk. In order to be sure that they operate with the correct run parameters (sourced from the shared memory) both programmes must be launched by the control task and are prevented from being launched directly by the user. To avoid conflicts on the bus, a lock-file forbids the running of two copies of these programmes (but a conscious user can circumvent this lock).

The principle of operations is trivial. When `daqd` is started, it initializes the hardware calling in turn the routines: `dvc01_start`, `dvc02_start`, ..., then it spawns `daqwrt` and begin to wait. As `daqwrt` starts, it initializes the output file and the goes to sleep.

When a SIGINT signal — as consequence of a SRQ on the bus — is dispatched to `daqd` it acquires the next free buffer and fills it calling in turn: `dvc01_read`, `dvc02_read`, ..., then issues a SIGURG signal. In response to this `daqwrt` awakes and starts to empty all filled buffers it can find; when no more filled buffers exist `daqwrt` goes back to sleep.

The cycle repeats untill a SIGTERM signal is recognized by `daqd`. Then it calls in turn the routines `dvc01_end`, `dvc02_end`, ..., and forwards the same signal to `daqwrt` that flushes the residual buffers and close the output file.

the user control

In the present version the two functions, *remote and local control*, shown in fig. 2 are accomplished by an unique, local, task. Consequently the user must `telnet` to `gnoA` and invoke: `/GNO/bin/daqctl`.

To master the variety of terminals, this programme relies only on the proper setting on the number of rows (minimum 24) and columns (minimum 80) and on the capability of addressing the cursor; practically any real or emulated terminal can be used. Graphics can be coarsely shown on the screen of alphanumeric terminals (using a VT100 emulator with 132 coluns and 58 rows it is not so bad); with more accuracy and better look on Tektronix 40xx or 41xx terminal or emulators; on REGIS terminals (e.g Digital VT230, VT330, VT430) and on any X11 terminal.

To avoid assumptions about the terminal used, the user interface appear like a hierarchical menu tree, driven by keystrokes (usually up and down arrows and enter keys). Instead, for to keep it simple, the implementation is realized with a flat bundle of coroutines and the depth of the menu routine calling is never greater than two.

Obviously many copies of `daqctl` can be active at the same time to look at the data acquisition progress; but only one copy at a time can controll (start / stop) the run. One by one, users can be allowed or prevented from becoming the *privileged* one.

hardware dependencies

In the present version any modification to the hardware obliges to modify few routines and to recompile and link the programme. Modules to be modified are: `/GNO/bin/devices.c` and, may be, `/GNO/bin/ctllib.c`

For the *nn*-th device to be serviced in `devices.c` there should be three routines:
– `dvc`*nn*`_start` that sends the proper sequences of GPIB commands to initialize the device;
– `dvc`*nn*`_read` that: i) sends the sequence to start the data trasmission from the device, ii) puts the received data in the right place in the event buffer, iii) updates the pointer to the data buffer;
– `dvc`*nn*`_end` that sends the sequence to terminate the device.

In `ctllib.c` there are two routines `dlg_dumpData` and `dlg_plotData` that depends on the actual meaning and format (word size, MSB first, ...) of the data and therefore could have to be adapted.

runs, data and all that

Runs are classified in four *types*: `fake`, `test`, `cali`-bration and `prod`-uction (*(b)* in table 4) and are identified by a progressive *run number* (*(a)* in table 4) that is increased at each run, irrespective of the run type. For each type the progressive number is recorded (*(c)* in table 4). By default `fake` runs doesn't write data on disk while the others will do; this default can be overriden by the operator.

The raw data are written on disk in the directory `/GNO/rawdata`. The file naming schema is: `runxxxx.ttttzzzz` where `run` is a fixed keyword; `xxxx` is the four digit run number; `tttt` is the four letter run type and `zzzz` is the four digit run number of this type.

Each file starts with a *start-of-run* block formed by:
– the number of bytes that will follow
– four byte identification (`SOR.`)
– all bytes from *(a)* to *(d)* of table 4.

The start-of-run block is followed by zero or more *event* blocks formed by:

– the number of bytes that will follow
– four byte identification (`EVT.`)
– all bytes from *(g)* to *(h)* of table 4.

Last block is the *end-of-run* block formed by:
– the number of bytes that will follow
– four byte identification (`EOR.`)
– all bytes from *(e)* to *(f)* of table 4.

testing

The system is exercised using the simple apparatus shown in fig. 4. A pulse from the pulse generator (PG) is sent to a discriminator (DSC) and (via an 80 ns delay) to a transient recorder (TD). The output of the discriminator: is counted on the scaler (SCA1), starts a 200μs gate generator (GG) and is feed to the coincidence.

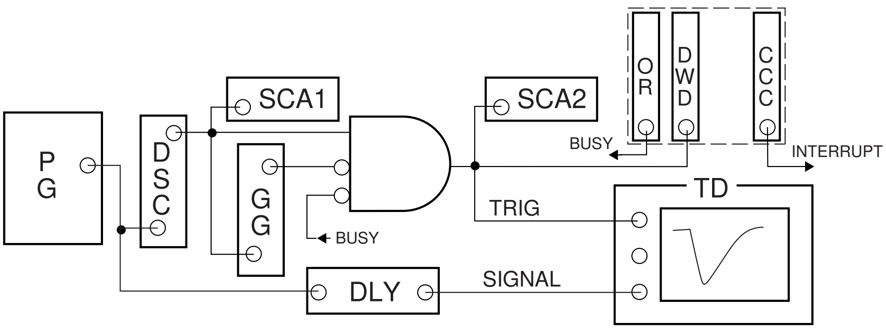

Fig. 4 — Schema of the test equipment

The output of the coincidence: (DSC) $\cdot$ ($\overline{\text{GG}}$) $\cdot$ ($\overline{\text{BUSY}}$) is counted on the scaler (SCA2); triggers the (TD); and is sent to a CAMAC dataway display (DWD) to produce a LAM; in respose, the crate controller (CC) produces a SRQ on the GPIB bus and this will send the SIGINT signal to `daqd`. The very first reaction is to turn-on the output register (OR) for to assert the *BUSY* line.

With reference to fig. 5, the dead time is given by the sum of the *reaction* time and the *service* time. The measured reaction time is $t_r = 130\mu$s; while the service time ranges from a minimum of $t_s = 1.1$ ms (only clear the LAM, read the scalers, turn-off the output register) to a maximum of $t_s = 3$ s (due to the conversion time of the TD).

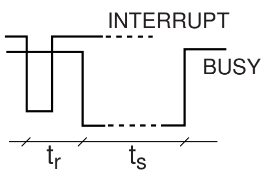

Fig. 5 — Timing