E10-2012-32

A. Yu. Isupov

NEW SOFTWARE OF THE CONTROL AND DATA ACQUISITION SYSTEM FOR THE NUCLOTRON INTERNAL TARGET STATION

Исупов А. Ю. E10-2012-32

Новое программное обеспечение системы управления и сбора данных
для станции внутренних мишеней нуклотрона

Создано новое программное обеспечение системы управления и сбора данных для станции внутренних мишеней (ITS) нуклотрона. Использована инфраструктурная система *ngdp* под Unix-подобной операционной системой FreeBSD, что позволяет по ходу сбора данных с ITS передавать их через компьютерную сеть, а также удаленно управлять внутренней мишенью.

Работа выполнена в Лаборатории физики высоких энергий им. В. И. Векслера и А. М. Балдина ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2012

Isupov A. Yu. E10-2012-32

New Software of the Control and Data Acquisition System
for the Nuclotron Internal Target Station

The control and data acquisition system for the Internal Target Station (ITS) of the Nuclotron (LHEP, JINR) is implemented. The new software is based on the *ngdp* framework under the Unix-like operating system FreeBSD to allow easy network distribution of the on-line data collected from ITS, as well as the internal target remote control.

The investigation has been performed at the Veksler and Baldin Laboratory of High Energy Physics, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2012

Dedicated to the bright memory of V. A. Krasnov

1. MOTIVATION AND MARKUP

The current version of the Nuclotron Internal Target Station (ITS) is described in [1]. The stepper motor and microstepper driver are still the same after the earliest ITS version [2].

During the 2010–2011 years the ITS control system was reimplemented to achieve the following goals:

- replacement of outdated DOS software, which does not support either the network, or the underlying computer hardware;
- replacement of the specific (almost unique now) CAMAC modules by the more generic ones with higher availability and repairability;
- integration of the already implemented ***targinfo(1)*** server, which as a workaround was used separately (under the SCAN DAQ [3]) from the internal target DOS software.

The present paper is focused on the software of the new ITS control and data acquisition (IntTarg CDAQ) system. The CAMAC hardware being used now by the IntTarg CDAQ is shown in Fig. 1, where we can see JINR manufactured generic modules only.

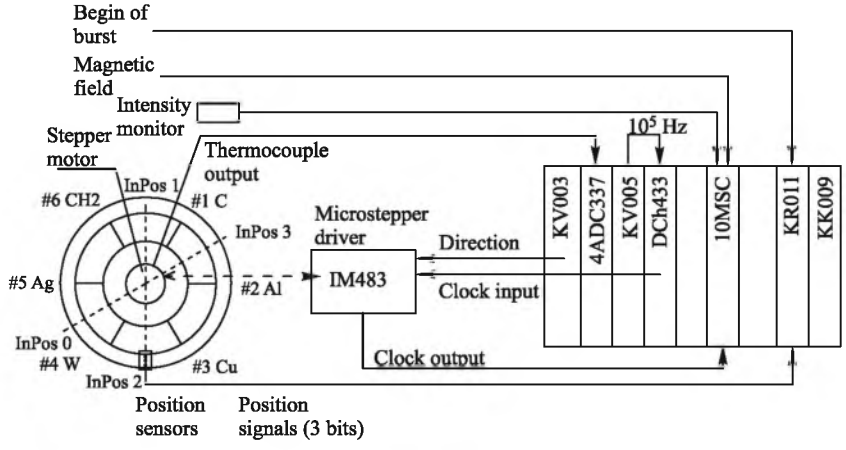

Fig. 1. Functional scheme of the internal target control

Through the present text the file and software package names are highlighted as *Italic text*, C and other languages constructions and reproduced «as is» literals — as `typewriter text`. Reference to the manual page named «qwerty» in the 9th section is printed as ***qwerty(9)***. Note also verbal constructions like «***accept(2)***ed» and «`mkpeer`ing», which mean «accepted by ***accept(2)***» and «peer making by `mkpeer`». Subjects of substitution by actual values are enclosed in the angle brackets: `<cnts_mask>`, while some optional parameters are given in the square brackets: `[-b<#>]`.

2. IntTarg CDAQ SOFTWARE

2.1. *ngdp* Based Design. As was noted in [4], the *ngdp* framework could be used to organize and manage the data streams originated, in particular, from the CAMAC hardware. To reach some independence on the CAMAC crate controller type, we use the current version of the *camac* package [5]. The *ngdp* using allows us to eliminate intermediate data storage on slow media like hard disks (HDD), as well as to gracefully distribute the data acquisition system between more than one networked computers if needed [6]. Of course, the *ngdp* usage essentially reduces the implementation efforts, as is shown in Subsec. 2.2.

In the presented design of the user context utilities we adhere to the *ngdp* and its predecessor *qdpb* [7] convention when some command-line flags have the same meanings for many utilities, as follows:

`-l` Write logging information by ***syslogd(8)***, facility `LOG_LOCAL0` (may be changed while compiling), levels `LOG_ERR` and `LOG_WARNING` instead of the standard error output.

`-v` Produce verbose output instead of the short one by default.

`-b<#>` Deal with the module, attached to the `#`th branch, instead of the 0th by default.

`-p<pidfile>` At startup write the own process identifier (PID) in `<pidfile>`. `-p-` means to use a compiled-in default for `<pidfile>`.

`-h` Write the utility usage to the standard error output and exit successfully.

So, in the specific utilities description we will not mention these flags. Note also, that each utility exits 0 if it is successful and nonzero — if an error occurs.

2.2. Ready Modules Used by IntTarg CDAQ. In the IntTarg CDAQ design we use the following already implemented entities (introduced by the *ngdp* framework, if not stated otherwise):

- The ***ng_ksocket(4)*** and ***ng_socket(4)*** node types are standard in the ***netgraph(4)*** package.
- The ***ng_camacsrc(4)*** node type (see [4]) allows us to inject data packets from a CAMAC interrupt handler (see 2.3.1) into ***netgraph(4)*** as data messages.

• The ***ng_fifos(4)*** node type (see [4]) implements the «selfflow»* queue with First Input First Output (FIFO) discipline and is able to:

† spawn `listen()`ing ***ng_ksocket(4)*** at startup;

† spawn `accept()`ing ***ng_ksocket(4)***(s) at each connection request from the known host(s) / port(s) up to the configured maximum, and/or

† accept hook connection from the local ***ng_socket(4)***(s);

† emit each data packet obtained on the `input` hook as soon as possible (ASAP) through all `accept()`ing ***ng_ksocket(4)***(s) and local ***ng_socket(4)***(s) currently connected;

† close `accept()`ing ***ng_ksocket(4)*** at EOF notification obtaining or connection loss.

• The ***ngget(1)*** (see [4]) is a utility for the packet stream extraction from ***netgraph(4)*** (usually through the ***ng_socket(4)*** node type).

• The ***writer(1)*** is a utility for packet stream writing into regular files on HDD, and introduced by the *qdpb* system [7].

2.3. Modules Specific for IntTarg CDAQ. *2.3.1. CAMAC Kernel Module* ***inttarg(4)***. The `inttarg` module is intended to work with the ITS CAMAC hardware, complies with requirements of the ***camacmod(9)*** and ***ng_camacsrc(4)***, so, it contains the CAMAC interrupt handler function. This handler recognizes the following interrupt occurrences (events):

• begin of burst (BoB),

• target arrival to the initial position (InPos), and

• target departure from it.

In total 8 packet types `INTTARG_CYC_{BEG,END,END[123]}`, `INTTARG_INF_0`, and `INTTARG_RUN_{BEG,END}` are produced. Note the `INTTARG_DAT_0` is not produced at all, because trigger events are absent in the present design. All 4 `END` packets have a variable length, while all the others — the fixed one.

At each BoB occurrence the `inttarg` produces the `INTTARG_CYC_BEG` packet, which contains at least the time stamp (`struct timeval`) of the BoB.

If the target should be active during the current burst, the corresponding (per-quantum) ***callout(9)*** handler is established to be executed once per our time quantum (compiled-in default is 10 ms, usually it equals to 10 OS ticks). After each quantum this handler increments the index in the trajectory description array already supplied by ***itoper(8)***, reads the member at this index, resets the effective microstep frequency and direction according to this member value, collects the experimental data from the ADC, and reestablishes itself. After the final quantum the per-quantum handler produces the `INTTARG_CYC_END2` packet (ADC data) and wakes up the kernel thread ***kthread(9)*** to read the memory buffer of

*Without internal bufferization and therefore request-free.

the 10MSC multiscaler and produce the `INTTARG_CYC_END1` (magnetic field from the 9th 10MSC up/down input), `INTTARG_CYC_END3` (the 0..7th 10MSC regular inputs) and `INTTARG_CYC_END` (target trajectory from the 8th 10MSC up/down input) packets. Note that `INTTARG_CYC_END` packet always is the latest data packet of the current burst, while `INTTARG_CYC_END[123]` are optional (depending on the software configuration).

The `INTTARG_CYC_END` packet contains up to 1+`arr_size`+1 of `int16_t` values. The `arr_size` is a basic size of internal arrays in the `inttarg` and configured by ***itconf(8)***, valid values are: 100..500, by default 500. The first (0th) `int16_t` value contains the `union inttarg_cyc_end_qdt` (see *einttarg.h*), which has the `quaval`, `dtype` and `tnum` fields. The `quaval` is a time quantum duration (10 ms), the `dtype` is a data type (valid values are `INTTARG_CYC_END_{INACTIVE,MKSTEPS,1_10MM,INVALID}`, see *einttarg.h*), and the `tnum` is a current target number (1..6). For our case of the active target the `dtype` is equal to `INTTARG_CYC_END_MKSTEPS` (or `1_10MM`, if the `inttarg` is compiled without 10MSC support and the reported trajectory is calculated instead of the really read-out one). Each other `int16_t` value is a signed microstep's number of the stepper motor during the corresponding time quantum. The positive values mean the movement from InPos, the negative ones — to InPos.

The `INTTARG_CYC_END1` packet contains up to 1+`arr_size`+1 of `int16_t` values. The first (0th) is the `union inttarg_cyc_end1_qf` (see *einttarg.h*). Each other `int16_t` value is a signed magnetic field difference during the corresponding time quantum. Naturally, the positive values correspond to the field increasing, the negative ones — decreasing.

The `INTTARG_CYC_END2` packet contains up to 1+`ADC_CHANS`∗(`arr_size`+1) of `uint16_t` values. The first (0th) equals to `ADC_CHANS` constant (= 4, means the number of ADC channels, numbered from the 0th to the 3rd). After the first `uint16_t` the `ADC_CHANS` `uint16_t` values represent the 1st time quantum, next `ADC_CHANS` `uint16_t` values — the 2nd time quantum, etc., and `ADC_CHANS` `uint16_t` values for the `arr_size`th final quantum. The 1st ADC channel is modified to obtain the thermocouple output to control the stepper motor temperature. The ADC value to temperature conversion is as follows: $T\,(^\circ\mathrm{C}) = -0.625\times$`ADC`$+628.75$.

The `INTTARG_CYC_END3` packet contains up to 1+`msc10nch`∗(`arr_size`+1) of `uint32_t` values. The first (0th) is the `union inttarg_cyc_end3_cnt` (see *einttarg.h*), which contains, in particular, the `msc10nch` and `msc10mask` fields. The `msc10nch` is a number (valid are 0..8) and the `msc10mask` is an 8-bit mask (valid are 0..`0xff`) of the used 10MSC regular inputs. The mask could be configured by ***itconf(8)***. After the first `uint32_t` the `msc10nch` arrays with the same length (up to `arr_size`+1 members) contain the `uint32_t` values of the 10MSC counts.

If the target should be inactive during the current burst, the end-of-burst (EoB) ***callout(9)*** handler is established to be executed after `arr_size` of 10 ms time quanta. The EoB handler produces only the `INTTARG_CYC_END` packet with an empty trajectory. This means that the packet contains the `union inttarg_cyc_end_qdt` only, where the `dtype` is equal to `INTTARG_CYC_END_INACTIVE`.

The `INTTARG_RUN_BEG` and `INTTARG_RUN_END` are produced at `start` and `stop` user requests (see ***itoper(8)***) and contain the time stamps (`struct timeval`).

The `INTTARG_INF_0` packet is produced to indicate the operation in progress or some error or warning condition, and it should be interpreted by the receiver (for example, ***itGUI(1)*** utility). The `INTTARG_INF_0` packet contains: `int32_t` value of the operation code (see *toper_op.h*), `int16_t` value of the error or warning code (see *inttarg_err.h* and ***errno(2)***), and `int16_t` value of the attribute (for example, the current target number at the `WARN_CHTARG` warning).

The `inttarg` module can be configured by the ***itconf(8)*** and controlled by the ***itoper(8)*** / ***itGUI(1)*** utilities. The `inttarg`'s `oper()` call supports at least the subfunctions enumerated in the ***itoper(8)***'s synopsis (see Subsubsec. 2.3.2). Generally speaking, the corresponding operations have the essentially asynchronous nature, so, we implement some kind of the finite state machine. This machine transits between well-defined states as a result of these operations execution. First of all, each operation should be added by the `oper()` into the FIFO queue (if a well-defined operation order permits it after the last already added operation). The queue is implemented by the singly-linked tail queue `STAILQ` (see ***queue(3)***), and the operation is represented in it by the `struct op_entry` (see *toper.h*). Each successfully added operation will be executed. Note that addition and execution are performed simultaneously with ***mutex(9)*** locking arbitration. Execution is completed in two phases: execution itself (i.e., some work with CAMAC) and asynchronous finish. The execution phase is performed by the `oper()` (if queue contains this command only), or by the kernel thread (after finishing the previous operation). The finish phase is made by the kernel thread waked up by IRQ handler, EoB or per-quantum ***callout(9)*** handler, or by own timed-out ***sleep(9)***. Operation can have one or more repetitions. So, `cycles #` operation has `#` repetitions, while the `targon` one is continuous (up to the `targoff` operation appearing in the queue). The operation execution and/or finishing failures lead to the queue discarding and the finite state machine appearing in the «unknown» (non-initialized) state.

The CAMAC hardware description and handling are separated from the `inttarg` module's source and grouped together in the single header *inttarg_hardware.h* . This header uses macro interface ***kk(9)*** specific for the KK009 crate controller [8] instead of the crate controller independent interface ***camac(9)***, because

the former interface allows us to slightly reduce the overhead for each CAMAC cycle [9].

*2.3.2. Configuration **itconf(8)** and Control **itoper(8)** Utilities*

```
itconf [-l] [-v] [-f<pflag>[,<pflag>...]] [-s{<arr_size>|-}]
   [-m<cnts_mask>] [-a] [-d{<driver>|-}] <module>
itconf -t [-l] [-v] <module>
```

In the first synopsis form the *itconf* utility configures the specified module `<module>` (in our case usually `inttarg`, see Subsubsec. 2.3.1 and ***inttarg(4)***) for work with driver `kk0` by default and produces packets with `F_CRC|F_TIME` (`#defined` in the *packet.h*) flags by default.

In the second synopsis form the *itconf* utility tests the configuration of the specified module `<module>` and writes it to the standard error output.

The default behavior of *itconf* may be changed by the following options:

`-d<driver>` Configure module for work with driver `<driver>` instead of default `kk0`. `-d-` means to use the compiled-in default for the driver. The default driver name may be changed at *itconf* compile time.

`-f<pflag>` Set `header.flag` field in the `make_pack()` produced packets in accordance to the `<pflag>` supplied. Valid values are: «`crc`», «`time`», «`none`» (see ***packet(3)*** for more details).

`-s<arr_size>` Set the size (number of members) of the arrays, which accumulate some statistics during the burst, to `<arr_size>`. Valid values are: 100..500 (corresponds to the burst of 1..5 s). `-s-` means to use the compiled-in default for `<arr_size>` (500), which can be changed at *itconf* compile time.

`-m<cnts_mask>` Set the bit mask for the 10MSC counter regular inputs, which will be read from CAMAC by configured module `<module>`. The `<cnts_mask>` value means as follows: the 0th nonzero bit marks the 0th counter input to be used, the 1st bit — the 1st input, etc., up to the 7th bit for the 7th input. `-m` absence in the command string leads to using the compiled-in default for `<cnts_mask>` (`0xff`, means — to use all 8 inputs), which can be changed at *itconf* compile time.

`-a` Request to read ADC0..3 at each time quantum and produce `INTTARG_CYC_END2` data packets with corresponding data. By default (without `-a`) the ADC1 is still being read at EoB or Final Quant and reported at `INTTARG_CYC_BEG` packets.

```
itoper [-l] [-v] [-b<#>] init|finish|start|stop|targon|targoff
    |status|cntcl|exec|done|clean|print
itoper [-N<#>] [-l] [-v] [-b<#>] targon
itoper -C<#> [-N<#>] [-l] [-v] [-b<#>] cycles
itoper -T<#> [-l] [-v] [-b<#>] chtarg
itoper [-r{<infile>|-}] [-l] [-v] [-b<#>] [-A<amax>] [-V<vmax>]
    [-c{-|<limsfile>}] settrj
itoper [-s{<outfile>|-}] [-l] [-v] [-b<#>] gettrj
```

In all the synopsis forms the *itoper* performs `oper()` call (see [4] and ***camacmod(9)***) with subfunction `fun` (see *inttarg.h*), defined by the first supplied argument, on the CAMAC module (usually ***inttarg(4)***) attached to the 0th branch, and writes the report about that action to the standard error output. The `init`, `finish`, `start`, `stop`, `targon`, `targoff`, `cycles`, `chtarg`, `settrj`, `gettrj`, `status`, and `cntcl` are `fun`s for production usage and expected to have self-explained names. Note with `-v` the *itoper* also uses `oper()` call with `status` subfunction. The *itoper* may be used, for example, to implement some commands in the supervisor configuration file ***sv.conf(5)*** (see Subsubsec. 2.3.3).

The default behavior of *itoper* may be changed, in particular, by the following options:

`-C<#>` This mandatory flag supplies the number of cycles `<#>` to be serviced by the internal target before the implicit stop (the third synopsis form of the *itoper*).

`-N<#>` If this optional flag is supplied, the one (last) of each `<#>` cycles will not be serviced by the internal target — so-called «drop each Nth cycle» mode (the second and third synopsis forms of the *itoper*). This allows another beam activity, for example, slow extraction. At the beginning of each burst previous to the inactive one the packet of `INTTARG_INFO_0` type with `WARN_PREP2DROP` value will be generated.

`-T<#>` This mandatory flag supplies the internal target number `<#>` to be made active (the fourth synopsis form of the *itoper*). Valid values are 1..6.

`-r<infile>` This optional flag supplies the `<infile>` name of the input file which contains the internal target trajectory to be programmed (the fifth synopsis form of the *itoper*). `-r-` means to use the compiled-in default for input file name (*$NGDPHOME/trj/in.trj*), which can be changed at *itoper* compile time. The same is used if the `-r` is absent.

`-s<outfile>` This optional flag supplies the `<outfile>` name of the output file where the current internal target trajectory should be stored (the sixth synopsis form of the *itoper*). `-s-` means to use the compiled-in default for output file name (*$NGDPHOME/trj/save.trj*), which can be changed at *itoper* compile time.

`-A<amax>` Sets the acceleration upper limit (in mksteps/ms^2) for the trajectory calculation to the supplied `<amax>` value. Default is 0.025.

`-V<vmax>` Sets the velocity upper limit (in mksteps/ms) for the trajectory calculation to the supplied `<vmax>` value. Default is 5.0.

`-c<limsfile>` Requires to read at startup the multipliers for the acceleration and velocity limits from `<limsfile>` (the fifth synopsis form of the *itoper*). `-c-` means to use compiled-in default for `<limsfile>` (*$NGDPHOME/etc/itGUI_lims.cfg*), which can be changed at *itoper* compile time. The same is used if the `-c` is absent. If the limits file opening or reading fails, the multipliers are 1.0 for the whole time range (0..5000 ms).

The `<infile>` and `<outfile>` contain the pair of ASCII float numbers delimited by space and/or tab symbol(s) per each line. (Lines are delimited by the newline symbol as is usual for UNIX textual files.) The first number is the time in ms (abscissa), the second — a rotating angle in arc degrees (ordinate). Lines with comment symbol «#» in the first position are ignored. For example, the requested trajectory in Fig. 3 is as follows:

```
#time   position (degrees)
0       0
900     33.2
3300    35.0
4300    0
```

Each line of the `<limsfile>` contains the four fields delimited by space and/or tab symbol(s). (Lines are delimited by a newline symbol as is usual for UNIX textual files.) The first field is a keyword, the `alim` means the line belongs to the acceleration's limitation, the `vlim` — to the velocity's one. The fourth field is an ASCII float and represents the multiplier for the limit, which is defined by the `-A/-V` option or by default. The second and third fields are ASCII integers from 0 to 499 and represent the lower and upper boundaries of the trajectory range (in the 10 ms units), where the multiplier will be applied. Lines with comment symbol «#» in the first position are ignored. For example, the requested trajectory in Fig. 3 uses the following limit multipliers:

```
#type   min     max     mult
alim    0       50      7.0
alim    50      200     0.3
vlim    0       150     1.2
```

*2.3.3. The **itGUI(1)** User Control Utility*

```
itGUI [-l] [-v] [-b<#>] [-a] [-L] [-M<mask>|-m<mask>] [-A<amax>]
   [-V<vmax>] [-r{infile|-}] [-s{outfile|-}] [-p{-|<pidfile>}]
   [-c{-|<limsfile>}]
itGUI -o [-l] [-v] [-b<#>] init|finish|start|stop|targon
   |targoff|status|cntcl|exec|done|clean|print
itGUI -o [-N<#>] [-l] [-v] [-b<#>] targon
itGUI -o -C<#> [-N<#>] [-l] [-v] [-b<#>] cycles
itGUI -o -T<#> [-l] [-v] [-b<#>] chtarg
itGUI -o [-r{<infile>|-}] [-l] [-v] [-b<#>] [-A<amax>] [-V<vmax>]
   [-c{-|<limsfile>}] settrj
itGUI -o [-s{<outfile>|-}] [-l] [-v] [-b<#>] gettrj
```

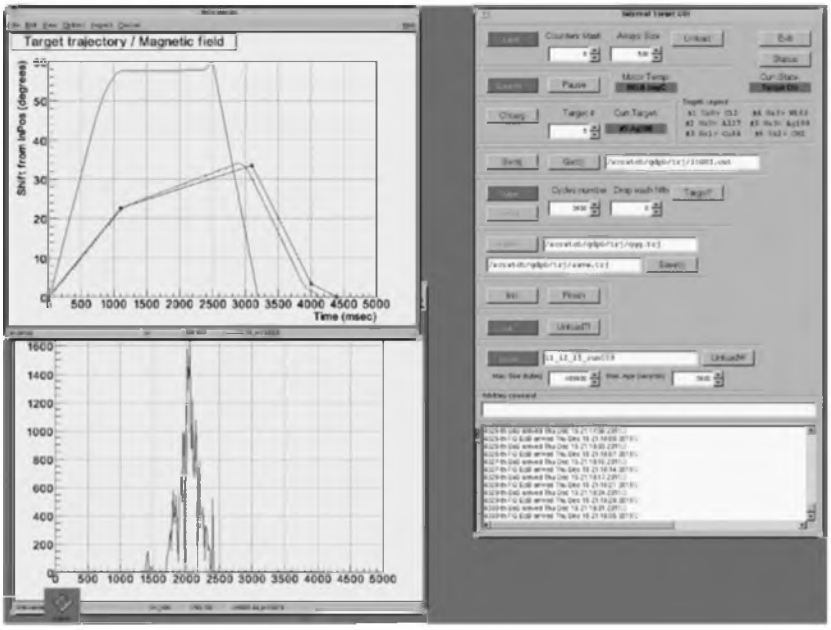

Fig. 2. The ***itGUI(1)*** screenshot. See the text for description

The *itGUI* provides the graphic user interface (GUI) for conversation with the IntTarg CDAQ system as well as for the graphic representation of the read out experimental data using the ROOT framework [10] libraries. In the first synopsis form the *itGUI* draws one main window of buttons (in Fig. 2 — the right window «Internal Target GUI») and some additional windows to display target trajectories (the upper left window) and other acquired data: magnetic field (the same window), multiscaler input(s) (the lower left window) and ADC channels (not shown). After that the *itGUI* tests the current IntTarg CDAQ state to highlight buttons, correspondingly, launches the child process to read *ngdp* packets from the standard input (usually supplied by the ***ngget(1)***), and goes into the endless loop (`TApplication::Run()`) of the graphic events handling. Note the *itGUI* could be safely and consistently restarted at any time during the IntTarg CDAQ working without the latter state changes.

If the *itGUI* called with the `-o` flag or under the *itoper* name, it behaves the same way as* ***itoper(8)*** utility in the corresponding synopsis forms (without `-o`, of course), see Subsubsec. 2.3.2.

*The ***itGUI(1)*** shares the corresponding source with the ***itoper(8)***.

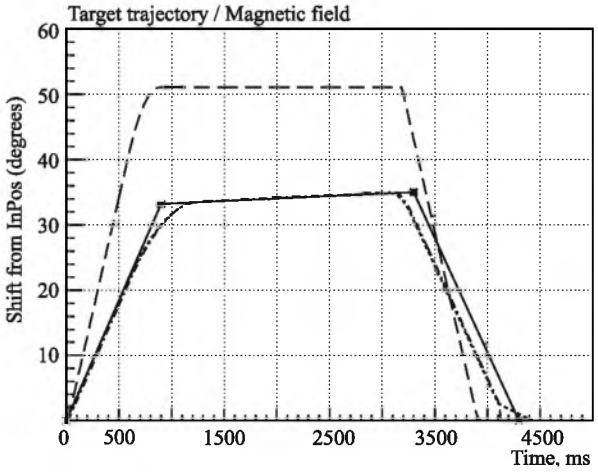

Fig. 3. The ***itGUI(1)*** trajectories window. See the text for description

The default behavior of *itGUI* may be changed, in particular, by the following options*:

`-a` Handle ADC0..3 histograms instead of ADC1 channel only for the stepper motor temperature calculation by default.

`-L` Leave mode — to survive after the data reading child process termination (usually after EOF obtaining while the offline data file reading). It allows us to play with the graphic output without future redraws.

`-M<mask>` Supplies the 8-bit mask `<mask>`, whose bits correspond to 10MSC regular inputs. Each nonzero bit means the corresponding counter to be additive during the current run instead of the counter resetting per each cycle by default.

`-m<mask>` The same as `-M`, however, it normalizes additive counters to the packet's number.

The *itGUI* is implemented with having in mind the supervisor utility concept (see [7]). According to this concept the *itGUI* has the configuration file (named by default *$NGDPHOME/etc/inttargsv.conf*) in the ***sv.conf(5)*** format (really Makefile, see also ***make(1)***). This file establishes the correspondence between the user commands («targets» in ***make(1)*** terminology) and actions which should be performed. This textual file could be revised easily without the *itGUI* recompile.

The main ***itGUI(1)*** window (`TGMainFrame`) contains (see Fig. 2) the buttons (`TGTextButton`), the string (`TGTextEntry`) and number (`TGNumberEntry`)

*Options already described in Subsubsec. 2.3.2 are not mentioned here. For the `<infile>`/`<outfile>` and `<limsfile>` formats see Subsubsec. 2.3.2.

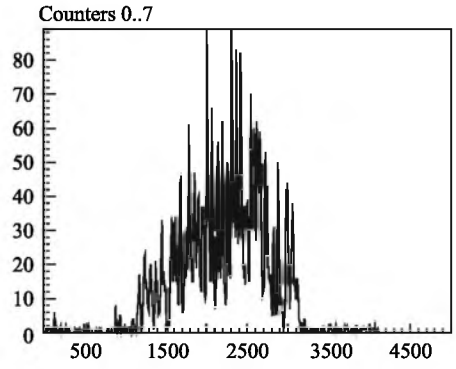

Fig. 4. The ***itGUI(1)*** counters window. See the text for description

input fields, the current state and target indication fields (`TGLabel`), and debug output viewer (`TGTextView`). Each button could be pressed by the left mouse button single-click, as well as the input focus could be placed into the input fields. The generic system startup direction is from up to down (and system stopping — from down to up). The buttons with «On» and «Off» meanings are placed in the same horizontal «engraved» frame from the left to the right.

Each button could be in the following states:

- active (could be pressed, black foreground);
- pressed (reverse shadow, grey foreground); or
- inactive (could not be pressed, grey foreground).

Each button could display the following operation states:

- the operation could be tried to perform (grey background);
- the operation successfully done (green background);
- the operation in progress or in queue (yellow background);
- the operation failed to be added to the queue or to be done (red background).

The buttons are:

`Load` loads and configures the ***inttarg(4)*** kernel module. The corresponding number input fields: `Counters Mask` — mask of the 10MSC input(s) to be read (0..`0xff`), `Arrays Size` — base array's size (500 per 5 s accelerator burst).

`Unload` (counterpart of previous) unloads the ***inttarg(4)*** kernel module.

`Exit` sends `SIGTERM` to the *itGUI*'s process group and exits the *itGUI*. Note the *itGUI*'s termination does not change the IntTarg CDAQ state in any way.

`Status` collects the status outputs from the number of system parts and displays these outputs in the debug output viewer.

`Readtrj` reads the requested trajectory from the input file under the name given in the string input field to the right from the `Readtrj` button. This reading

totally replaces the current requested trajectory, as well as its interactive modification, and leads to the calculated trajectory updating.

`Savetrj` saves the current requested trajectory in the output file under the name given in the string input field to the left from the `Savetrj` button.

`Continue` starts the data acquisition, i.e., handling of the BoB interrupts. In this state the system is ready for target walking, and the ***targinfo(1)*** server has the data to be distributed.

`Pause` (counterpart of previous) stops the BoB interrupts handling. In this state the ITS hardware could be safely powered off/on.

`Settrj` supplies the current calculated trajectory for the ***inttarg(4)*** kernel module.

`Gettrj` gets the calculated trajectory from the ***inttarg(4)*** kernel module and prints it in the debug file under the name given in the string input field to the right from the `Gettrj` button.

`Chtarg` sets the target with number `<#>` to be a current (ready to walk) target. The corresponding number input field: `Target #` — the number of the target to be the current one (valid numbers are 1..6). The correspondences between the numbers and materials are displayed to the right from the `Chtarg` group frame.

`Targon` allows the current target to walk during each cycle infinitely (up to explicit denying by the `Targoff` pressing). The target starts to walk at the nearest BoB. During the latest of each N bursts the target could be not inactive, if the $N \geqslant 2$ value is supplied with the corresponding number input field `Drop each` Nth. Valid numbers are 2..3600, and 0, 1, which mean that the target does not drop the bursts.

`Targoff` (counterpart of previous) denies the current target to walk. The target stops to walk after the nearest EoB or before the nearest BoB.

`Cycles` allows the current target to walk during each cycle of the nearest `#` cycles (or up to explicit denying by the `Targoff` pressing), supplied with the corresponding number input field `Cycles number` (valid numbers are 1..3600). The target starts to walk at the nearest BoB. During the latest of each N bursts the target could be inactive, if the $N \geqslant 2$ value is supplied with the corresponding number input field `Drop each` Nth. Valid numbers are 2..3600, and 0, 1, which mean that the target does not drop the bursts.

`Init` initializes ITS hardware, in particular, CAMAC modules in the read-out crate, and positions the target into the nearest InPos. Note the `Init` is also a part of `Load`.

`Finish` (counterpart of previous) de-initializes the internal target hardware. In the current design it is not needed at all.

`LoadW` loads the ***writer(1)*** — utility to write the packet stream into files on HDD. The corresponding input fields:
(string) — base name of the data files for the current ***writer(1)*** run;

`Max. Size (bytes)` (number) — the recommended size for each file;
`Max. Age (seconds)` (number) — the recommended age for each file.

`UnloadW` (counterpart of previous) unloads the ***writer(1)*** utility and consequently terminates the data writing of the current run.

`LoadTI` loads the ***targinfo(1)*** — server, which distributes the read out internal target trajectory to its already registered clients.

`UnloadTI` (counterpart of previous) unloads the ***targinfo(1)*** server.

Each but `Exit`, `Readtrj`, `Savetrj` button corresponds to the command (target) under the same name (without capitalization) in the supervisor configuration file, so the IntTarg CDAQ control has two functionally equivalent interfaces: graphic (by ***itGUI(1)***) and textual (by configuration file ***make(1)***ing). Note, however, that ***itGUI(1)*** itself uses the configuration file to perform the complex commands (`[un]load`, `[un]loadw`, `[un]loadti`, `status`) only. In contrast, the simple commands (see `-o` synopsis forms) corresponding directly to ***inttarg(4)***'s `oper()` subfunctions, are performed internally using the code shared with ***itoper(8)***.

The indication fields in the main window are:

`Curr.State:` (under the `Status` button) displays the current state of the operations queue (see Subsubsec. 2.3.1), one of: «Unknown» (on grey background — before `Load` and after `Unload`, on red — otherwise), «Init», «Start», «Stop», «Cycles», «Target On» (all on green).

`Curr.Target:` (to the right for the `Chtarg` button) displays the current target number and material (on the green background — after successful target change, on the yellow one — after startup and while the target changing, on the red — after the unexpected target change or after obtaining the invalid number of the current target), for example: «`#5 Ag108`».

The string input field named «Arbitrary command» (just above the debug output viewer) allows the user to perform any target from the supervisor configuration file.

The ***itGUI(1)*** data windows are as follow:

trj canvas The internal target trajectories — requested (in black, with asterisk points), calculated (in red) and read out (in green, only after cycles with the active target) from the 8th 10MSC up/down input — as well as the magnetic field (in blue) read out from the 9th 10MSC up/down input are displayed in the single `TCanvas` named «trj canvas». On the *itGUI* screenshot (Fig. 2) we can see this canvas as the left upper window frame with the «Target trajectory / Magnetic field» title. With the higher resolution such canvas is shown in Fig. 3. The calculated (dotted) and read out (dash-dotted) trajectories are in some segments below the requested trajectory (the solid line with two asterisks) and approximately coincide. The last segment of the read-out curve goes to zero vertically, because the multiscaler is not

read after the final time quantum, however, the target really goes into InPos with the fixed 1 mkstep/ms velocity, as is shown by the calculated curve. The arbitrary scaled magnetic field (dashed curve) is above all the others. The abscissa is the time in milliseconds, the ordinate is a target shift in arc degrees (InPos at 0°, beam center approximately at 30°). Note, the requested and calculated trajectories could not be very close to each other (as in Fig. 3), because the trajectory calculation algorithm has the upper limits for the target velocity (leads to the lower slope angle) and acceleration (leads the to angle smoothing). To be more flexible, these limits could be varied along the trajectory according to the configuration file *itGUI_lims.cfg* (format described in Subsubsec. 2.3.2). So, the trajectory in Fig. 3 is calculated with the following limits:
`amax`< 0.175 mksteps/ms^2 at 0..500 ms;
`amax`< 0.0075 mksteps/ms^2 at 500..2000 ms;
`vmax`< 6.0 mksteps/ms at 0..1500 ms
(in other time ranges the both limits are default). The requested trajectory could be read from the input file under the name given in the corresponding `TGTextEntry` by pressing the `Readtrj` (see above). Reading from the file totally replaces the current requested trajectory. The current requested trajectory could be interactively modified. The point could be added by the left mouse button double-click, while the existing point could be removed by the middle mouse button double-click (see also the hot keys below). Note, neither the points nor the whole trajectory should be moved. After each modification of the requested trajectory the calculated trajectory is updated. The current requested trajectory could be saved in the output file under the name given in the corresponding `TGTextEntry` by pressing the `Savetrj` (see above). The following hot keys are supported while mouse focus is in «trj canvas»:
a, A — add the point under the current mouse position;
d, D — delete the point under the current mouse position;
i, I — input the point using the `TGInputDialog` (the coordinates should be entered as the time-angle pair of the float numbers delimited by space and/or comma);
u, U — remove the last point added by the left double-click or hot keys A, a, I, i (could be used many times);
p, P — print the calculated trajectory into the debug file under the name given in the `TGTextEntry` to the right from the `Gettrj` (see above);
r, R — read out the requested trajectory from the input file under the name given in the `TGTextEntry` to the right from the `Readtrj` (see above);
c, C — clean out the trajectory (only (0,0) and (5000,0) points are preserved);

s, S — save the current requested trajectory in the output file under the name given in the `TGTextEntry` to the left from the `Savetrj` (see above);
T — supplies the current calculated trajectory for the ***inttarg(4)*** kernel module (the same as `Settrj`, see above);
t — gets the calculated trajectory from the ***inttarg(4)*** kernel module and prints it in the debug file under the name given in the `TGTextEntry` to the right from the `Gettrj` (see above);
q, Q — quit the *itGUI* (the same as `Exit` button pressing, see above).

cnts canvas From 0 up to 8 counter values, read out from the 0..7th 10MSC inputs, are represented by `TH1D`s and displayed in the single `TCanvas` «cnts canvas» iconified at startup. In Fig. 2 it is the lower left window frame with the single curve, which represents the interaction intensity monitor counts for the last processed cycle. This plot allows the user to tune the target movement trajectory. Numbers and positions of the inputs to be used are configured (see ***itconf(8)***) while loading of the ***inttarg(4)*** kernel module. The intensity monitor counts during the single cycle for the C target run is shown in Fig. 4.

cv_adc_N Four ADC channel histograms (`TH1`) are displayed each in its own window (`TCanvas`) «cv_adc_0»..«cv_adc_3» iconified at startup (in Fig. 2 we cannot see them), if the `-a` option is specified.

*2.3.4. The **targinfo(1)** Trajectory Server.* The ***targinfo(1)*** is a server, which provides its clients with the internal target trajectory data in each accelerator cycle at the end of this cycle.

`targinfo [-l] [-t] [-f<#>] [-p{-|<pidfile>}] [-a<address>[ ...]]`

In this synopsis form the *targinfo* listens to the TCP/IP socket on the host `he117-90.jinr.ru`, port 12345 to get client connection requests, and reads packets from the standard input. From each of the obtained `INTTARG_CYC_BEG` packets the *targinfo* collects the BoB timestamp, while from each of the `INTTARG_CYC_END` ones it collects the microstep number array of the ITS stepper motor. Once per cycle the *targinfo* writes the cycle timestamp and some target trajectory data (format described below) to all the currently connected clients (if any). Up to `CLIENTS_MAX` clients (usually 5) can be serviced simultaneously, connections closed by the peer are recycled.

The default behavior of the *targinfo* may be changed by the following options:

`-t` Exits at all negative conditions from ***packet(3)*** functions instead of the exit only at −3 by default.

`-f<#>` Assigns the supplied `<#>` number to the `pack_flags` variable for the ***packet(3)*** `read_pack()` function. `-f` absence in the command string leads to using the compiled-in default for `<#>` (`F_CRC` (see *packet.h*), because this is the only checkable value for reading).

`-a<address>` Restricts clients to connect from specified `<address>` only instead of allowing the client to connect from any one by default. `<address>` can be an Internet name in domain notation or IP address as four decimals separated by dots. Note that `-a` option can be specified with different `<address>`es up to `CLIENTS_MAX` times.

The data format preserved after workaround implementation under the DAQ system of the SCAN setup [3] is as follows:

• timestamp of the burst begin (8 bytes of the `struct timeval`);

• signed target shift in 1/10 mm per each 20 ms (the 250 `int16_t`s allows us to cover up to 5 s burst).

The format also indicates the following situations:

• A nonactive target. If the target was not injected into the beam during the current cycle, all the 250 `int16_t`s are equal to 0.

• Some erroneous target behaviour. If the target was walking incorrectly, all 250 `int16_t`s are equal to `SHRT_MAX` (`0x7fff`).

• The next cycle will be nonactive. The ITS control supports a special operation mode, in which the target remains inactive each Nth of N cycles to allow other beam activities. In this mode, if the trajectory data are `SHRT_MAX-1` (`0x7ffe`), the internal target will be inactive during the next cycle. Note, these data will be written soon after obtaining `INTTARG_CYC_BEG` packet in contrast with all the other data types generated at `INTTARG_CYC_END` arrival. Note also, that the normal trajectory output will be generated for the current cycle, too, and during the next (inactive for the internal target) cycle the trajectory data will be zero (as should be expected).

2.4. Bringing All Things Together. The *ngdp* graph used by the IntTarg CDAQ is shown in Fig. 5. It is very much like *ngdp*'s CAMAC Front-End Modules (FEM) level proposed in [6]. The node named `fifo:` of type `ng_fifos` is instantiated (`mkpeer`ed) at the `he117-90.jinr.ru` host bootstrap time using the script *$NGDPHOME/etc/fifo.ngctl* processed by the ***ngctl(1)***. During `ng_fifos` startup it `mkpeer`s the node of type `ng_ksocket` with automatically chosen name (`0x2_listen:` in Fig. 5), and `connect`s the remote hook `inet/stream/tcp` with its own hook `listen`. So, the `ng_fifos` and its `listen()`ing `ng_ksocket` are still ready from OS's boot to shutdown.

The `ng_fifos` provides identical packet streams through all the currently connected outputs, both the local and remote ones. Each output could be connected and disconnected without disturbing other outputs, so the packet stream consumers (***writer(1)***, ***targinfo(1)***, and ***itGUI(1)***) could be started and terminated independently of each other. These utilities are launched by the `loadw`, `loadti` and `loadgui` commands from the *$NGDPHOME/etc/inttargsv.conf* to be readers of the pipes, where the writers are the ***ngget(1)***s. Each ***ngget(1)*** `mkpeer`s the `ng_socket` instance (`ngget73583:` and `ngget73547:` in Fig. 5) and

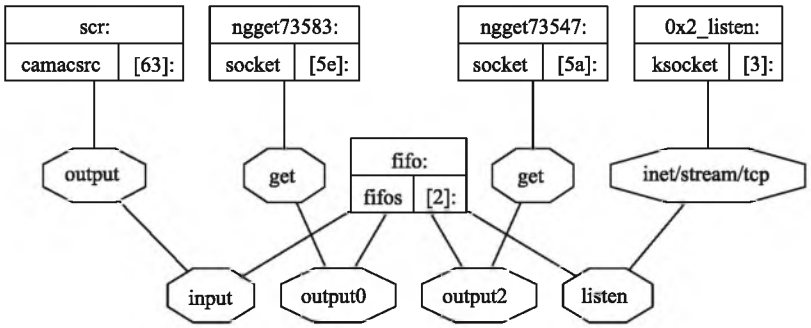

Fig. 5. The IntTarg CDAQ core is implemented by the *ngdp* graph. Rectangles are nodes with: name (up), type (left), ID (right); octagons are hooks named within. `ng_fifos` has two local output streams through `ng_sockets`, as well as `listen()`ing `ng_ksocket`

`connect`s it by hook `get` with `ng_fifos`'s hook `output<N>`. The simultaneously allowed numbers of both the `output<N>` hooks and `accept()`ing `ng_ksocket`s are the compiled-in parameters of `ng_fifos`.

The `load` command of the supervisor configuration file, in particular, loads the ***inttarg(4)*** interrupt handler and executes the ***ngctl(1)*** utility to proceed the script *$NGDPHOME/etc/camacsrc.ngctl*, which `mkpeer`s the node named `src:` of type `ng_camacsrc`. This node communicates (see [4] for details) with CAMAC kernel module ***inttarg(4)***, and `connect`s its own hook `output` with the hook `input` of the `ng_fifos`. After that the IntTarg CDAQ graph has all components which are provided by the current design.

The supervisor configuration file *$NGDPHOME/etc/inttargsv.conf* allows the user to control the IntTarg CDAQ in the command-line mode through a simple textual terminal (without GUI). Of course, the requested trajectory could not be corrected interactively in this mode and user cannot see all the read-out data. However, the trajectory file is textual (see Subsubsec. 2.3.2), so it could be edited easily.

The overall IntTarg CDAQ layout is pictured in Fig. 6, where we can see the host `he117-90.jinr.ru` as the rectangle entitled «IntTarg CDAQ». In the user context the three processes (***itGUI(1)***, ***targinfo(1)*** and ***writer(1)***) obtain three identical packet streams from three ***ngget(1)***s, which read three ***ng_socket(4)***s connected to ***ng_fifos(4)***. The packet streams with the same contents could be also transferred remotely through the ***accept(2)***ing ***ng_ksocket(4)***s instantiated after client's ***connect(2)***ion to the ***listen(2)***ing ***ng_ksocket(4)***. (The ***netgraph(4)*** behaviour mimics the BSD socket handling scheme.) In the present state we have no remote consumers of the full packet stream, however, they can appear in future (see Subsec. 2.5).

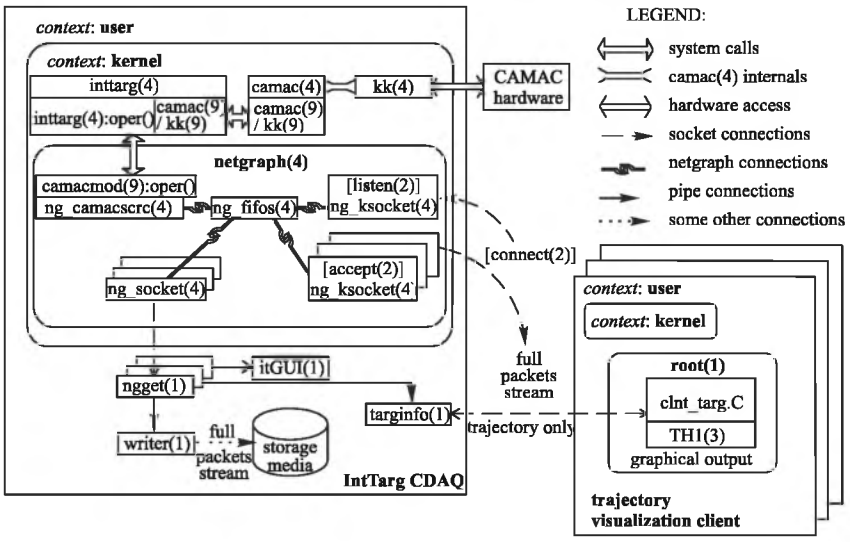

Fig. 6. Overall IntTarg CDAQ layout

The rectangle «trajectory visualization client» in Fig. 6 is another host which executes one of the possible clients of the ***targinfo(1)*** server — the ROOT script *clnt_targ.C* . Note the ***targinfo(1)*** distributes the read out trajectory only, not the packet stream (see also Subsubsec. 2.3.4). So, the ***targinfo(1)*** is preserved mostly for the backward compatibility and could be retired in future.

2.5. Nuclotron Run Experience and Future Directions. The IntTarg CDAQ was used to control the internal target during the March and December 2011 Nuclotron accelerator runs. The total beam time of the ITS operation was approximately 150 h on the deuteron beam at $T_{\text{kin}} = 250{-}500$ MeV/nucleon. In particular, the «drop each Nth cycle» mode was successfully used in March 2011.

The IntTarg CDAQ usage experience during the March 2011 Nuclotron accelerator run provides some hints to improve the software, first of all ***itGUI(1)***, in some aspects. So, the ROOT `TGraph` was replaced by the `TH1D` to represent most of the visualized curves with many (some hundreds) points since the `TGraph` has shown a dramatically low visualization performance (ROOT 5.16). Also the ROOT `TThread` was eliminated in favor of traditional UNIX child process `fork()`ing to separate the execution stream for the data packets reading from the main execution stream for the X11 events handling. This choice has been justified by the fact, that the BSD scheduling algorithm for processes instead of threads is more mature, refined, and featured. The `-a` option was added to the ***itGUI(1)*** to reduce visualization expenses by default, and visualize ADC

histograms with `-a`. If the same option is not supplied to ***itconf(1)***, the ***inttarg(4)*** module will be configured not to collect ADC data for histogramming, because these data are currently not useful. Only the stepper motor temperature channel (ADC1) will be read once per cycle (at EoB or final quantum). The corresponding value could be found at the end of `INTTARG_CYC_BEG` packet, which was enlarged by 2 bytes. This feature reduces the CAMAC activity overhead per each time quantum and allows one to watch the stepper motor temperature also during the cycles with an inactive target. All the mentioned software improvements were successfully tested during the December 2011 Nuclotron run.

The full IntTarg CDAQ system data set in the packet stream form could be provided on-line for clients on the remote hosts. A client could be something like a (sub)event builder ((Sub)EvB, see [6]) implemented in the kernel context by, for example, the following *ngdp* graph: `ng_ksocket`→`ng_defrag`→`ng_em[s]`. In the user context a client could be, for example, a pipe of the ***hose(1)*** utility (writer end) from the *netpipes* package (see ***netpipes(1)***) and a some read-only version of the ***itGUI(1)*** (reader end), which allows the user to observe but not to control the internal target. Of course, users are free to implement their own clients using the BSD socket and *ngdp* ***packet(3)*** program interfaces (APIs).

Some minor updates of the IntTarg CDAQ are also possible in future. A separate canvas for the trajectory changes and calculations during the target walking could be added. The magnetic field reading with the inactive target will be useful. The target trajectory calculation algorithm has some annoying features, so it could be revised more or less essentially. The 8th and 9th multiscaler input readings could be prolonged after the final quantum.

CONCLUSIONS

The new control and data acquisition system for the Nuclotron ITS has been implemented using the *ngdp*, *camac*, and ROOT packages to allow easy network distribution and integration. The outdated both DOS software and underlying CAMAC and computer hardware have been replaced. The previously implemented ***targinfo(1)*** server has been integrated into the IntTarg CDAQ now. During 150 h of the 2 Nuclotron runs the IntTarg CDAQ has demonstrated the operation stability and target manipulation convenience.

Acknowledgements. The author has a pleasure to thank S. G. Reznikov, who has designed and implemented the hardware upgrade of the ITS control scheme on the generic CAMAC hardware, V. P. Ladygin — for initiation of the presented developments, V. A. Krasnov — for useful discussions, S. M. Piyadin, A. N. Livanov and A. N. Khrenov — for support during the run. The present work was supported in part by the RFBR grant 10-02-00087a.

REFERENCES

1. *Anisimov Yu. S. et al.* A new construction of Nuclotron internal target station and its control system // Proc. of the International Workshop — Relativistic Nuclear Physics: from Hundreds of MeV to TeV, Stara Lesna, Slovakia, 2003. JINR, 2003. P. 117.
2. *Malakhov A. I. et al.* // NIM A. 2000. V. 440. P. 320.
3. *Afanasiev S. V. et al.* The data acquisition and trigger of the SCAN setup // Instr. and Experim. Techn. 2008. V. 1. P. 34–39 (in Russian).
4. *Isupov A. Yu.* CAMAC subsystem and user context utilities in *ngdp* framework. JINR Commun. E10–2010–35. Dubna, 2010. 20 p.
5. *Gritsaj K. I., Olshevsky V. G.* Software package for work with CAMAC in Operating system FreeBSD. JINR Commun. P10–98–163. Dubna, 1998. 16 p. (in Russian).
6. *Isupov A. Yu.* The *ngdp* framework for data acquisition systems. JINR Commun. E10–2010–34. Dubna, 2010. 20 p.
7. *Gritsaj K. I., Isupov A. Yu.* A trial of distributed portable data acquisition and processing system implementation: the *qdpb* — data processing with branchpoints. JINR Commun. E10-2001-116. Dubna, 2001. 19 p.
8. *Churin I., Georgiev A.* // Microprocessing and Microprogramming. 1988. V. 23. P. 153.
9. *Isupov A. Yu.* SPHERE DAQ and off-line systems: implementation based on the *qdpb* system. JINR Commun. E10-2003-187. Dubna, 2003. 17 p.
10. *Brun R., Rademakers F.* ROOT — an object oriented data analysis framework // Proc. of the AIHENP'96 Workshop, Lausanne, Switzerland, 1996. NIM A. 1997. V. 389. P. 81–86.

Received on March 21, 2012.

Корректор *Т. Е. Попеко*

Подписано в печать 14.06.2012.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,43. Уч.-изд. л. 1,93. Тираж 250 экз. Заказ № 57665.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@jinr.ru
www.jinr.ru/publish/