VME–based DAQ system for the Deuteron Spin Structure setup at the Nuclotron internal target station

Alexander Isupov[1,*]

[1]Laboratory of High Energy Physics, Joint Institute for Nuclear Research, Dubna 141980, Russia

Abstract. The new powerful VME–based data acquisition (DAQ) system has been designed for the Deuteron Spin Structure setup [1] placed at the Nuclotron Internal Target Station [2]. The DAQ system is built using the netgraph–based data acquisition and processing framework *ngdp* [3, 4]. The software dealing with VME hardware is a set of *netgraph* nodes in the form of the loadable kernel modules. The specific for current implementation nodes are described, while specific software utilities for the user context are the following. The *b2r* (binary–to–*ROOT*) server converts raw data into per trigger and per accelerator spill representations, which are based on C++ classes derived from the *ROOT* framework [5] ones. This approach allows us to generalize the code for histograms filling and polarization calculations. The *b2r* optionally stores *ROOT* events as *ROOT* `TTree` in file(s) on HDD, and supports the design of some express offline. The *histGUI* software module provides an interactive online access for human operator to histograms filled by the *r2h* (*ROOT*–to–histograms) server, which obtains the *ROOT* event representations from *b2r*. The *r2h* supports the calculation and histograming of runtime configurable variables as well as raw data variables, and optionally stores *ROOT* histograms in file(s) on HDD. Since the spin studies at the Nuclotron require fast and precise determination of the deuteron and proton beam polarization, the polarization calculator software module is introduced. This calculator based on runtime configurable *r2h* code allows us to compute polarization values online and integrate them into the Web–based scheme of representation and control of the polarimeters [6, 7].

1 Introduction

The new powerful VME–based data acquisition (DAQ) system has been designed for the Deuteron Spin Structure (DSS) setup [1], [8], [9], [10] placed at the Nuclotron Internal Target Station (ITS) [2]. The DSS studies the polarization observables in the dp elastic scattering for several years [11], [12], [13], [14]. Vector and tensor polarizations of the deuteron and proton beams from the LHEP–JINR polarized ions source (SPI) [15], as well as A_y, A_{yy}, A_{xx} analyzing powers of the reactions have been successfully measured in the beam energy range from 135 MeV/nucleon to 900 MeV/nucleon. The DAQ system which will be described in this report is built using the *netgraph*–based data acquisition and processing framework *ngdp* [3, 4]. The software dealing with VME hardware is a set of *netgraph* package nodes in the form of the loadable kernel modules to work in the kernel context of the operating

*e-mail: isupov@moonhe.jinr.ru

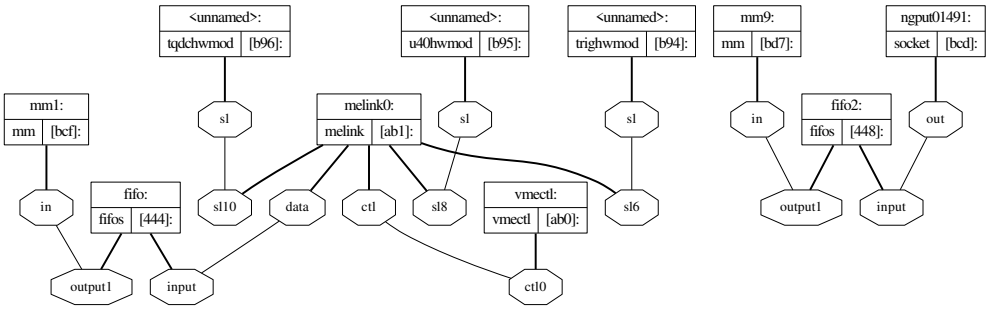

Figure 1. The simplified *netgraph* graph used by DSS DAQ system to deal with VME hardware and manage the produced data streams. Rectangles are nodes with: name (up), type (left), ID (right); octagons represent hooks with their names inside

system. The user context utilities implement Graphical User Interfaces (GUI) for interaction with human operator as well as data representation converters both are based on C++ classes derived from the *ROOT* framework [5] ones.

Through the presented text the file and software package names are highlighted as *italic text*, while constructions of C and other languages — as `typewriter text`. Reference to the manual page with name "qwerty" in the 9th section is printed as ***qwerty(9)***. All the mentioned trademarks are properties of their respective owners.

2 Dealing with VME hardware

The heart of DSS DAQ system — *ngdp* graph to deal with VME hardware and manage the produced data streams — is presented in Fig. 1. This picture was automatically generated by the `dot` command of ***ngctl(8)*** utility from the graph was really built, and manually simplified by stripping out both nodes non–essential for understanding and details of the too technical nature. The *ngdp* graph like any ***netgraph(4)*** graph works in the operating system (OS) kernel context and contains the nodes (rectangles in Fig. 1) connected by graph edges, which are produced by pairs of the so called hooks (octagons in Fig. 1). The data messages flow along the edges while the control messages are delivered to the corresponding nodes directly. For reasons to use the *netgraph* package as some data streams implementation in the OS kernel, see [3]. The nodes are implemented in the form of the loadable kernel modules (KLD), whose approach simplifies and essentially speeds up the debugging. The data processing code execution in the kernel context allows us to handle the data obtained from the VME master in the fastest possible way [3]: out of the preemptive scheduling scope and without data copying overhead.

The VME hardware handling scheme is as follows. Each kind of VME master (with embedded CPU or adapter card for computer interconnection) and VME hardware module is represented by the corresponding *netgraph* node type — VME driver and handler of VME hardware module (lets name it VME node below), respectively. Each node type should be instantiated as many times as needed to reflect the existing quantity of the corresponding hardware units.

The VME driver ***ng_melink(4)*** is a KLD module (named `melink0` in Fig. 1) intended to handle M–Link FVME/FVME2 VME master hardware [16], while the ***ng_vmedummy(4)*** one — imitates some VME master hardware for debug purposes. So the VME driver contains UNIX hardware driver with `probe()`, `attach()`, `detach()` procedures, interrupt handler,

kernel threads ***kthread(9)*** to serve requests ***queue(3)*** and process data in parallel, and is a *netgraph* node simultaneously.

The VME node is a KLD module in the *netgraph* style intended to deal with some VME hardware module (or imitate one for debug purposes as ***ng_genhwmod(4)*** do) by mediation of VME driver. Currently we support: the trigger TTCM/FVME2TMWR modules [16] with PCB 2.0/2.0 and firmware version 1.1.14136 (final) / 1.1.20532 – by the ***ng_trighwmod(4)*** node type; the trigger logic U40VE modules [16] with PCB 2.1 and DSS firmware version 2.0.22585 – by the ***ng_u40hwmod(4)*** node type; and the TQDC16 time– and charge–to–digital converter modules [16] with PCB 2.1, 3.0, 4.1 and firmware version up to 2.0.21992 – by the ***ng_tqdchwmod(4)*** node type.

Interconnections in the DAQ graph (see Fig. 1) are the following: VME driver in <M>$^{\text{th}}$ crate is connected by own `ctl` hook to `ctl`<M> hook of special control node ***ng_vmectl(4)*** node while VME node in <N>$^{\text{th}}$ slot — by own `sl` hook to `sl`<N> hook of VME driver of the corresponding VME crate.

The full data flow scheme of the DSS DAQ system is as follows:

$$\text{ng\_melink} \rightarrow \text{ng\_fifos} \begin{array}{l} \nearrow \texttt{ng\_mm} \not\rightarrow \textit{ngget} \mid \textit{writer} \\ \rightarrow \texttt{ng\_mm} \not\rightarrow \textit{ngget} \mid \textit{b2r} \mid \textit{ngput} \not\rightarrow \texttt{ng\_socket} \rightarrow \\ \searrow \texttt{ng\_mm} \not\rightarrow \textit{ngget} \mid \textit{b2r} \text{ (EoB counts)} \end{array}$$

$$\rightarrow \text{ng\_fifos} \begin{array}{l} \nearrow \texttt{ng\_mm} \not\rightarrow \textit{ngget} \mid \textit{r2h} \; -\!-\!-\!\rightarrow \textit{histGUI} \\ \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \backslash \; -\!-\!-\!\rightarrow \textit{histGUI} \\ \searrow \texttt{ng\_mm} \not\rightarrow \textit{ngget} \mid \textit{r2h} \text{ (polar. calc.)} \quad \cdots \end{array}$$

where:

$\rightarrow$	***packet(9)*** [17] data stream in the kernel context (along graph edges);
$\not\rightarrow$	context boundary crossing by the data stream;
$\mid$	***packet(3)*** [17] data stream in the user context (so called pipe);
$-\!-\!\rightarrow$	socket connection (here `TServerSocket` / `TSocket` from *ROOT* framework [5]);
$\cdots$	number of the same elements.

The *netgraph* (kernel context) part of this scheme can be seen in Fig. 1.

The ***ng_melink(4)*** VME driver node named `melink0` in Fig. 1 is the source of data packets stream which supplied to the ***ng_fifos(4)*** node [4] named `fifo` through the pair of hooks `data` and `input`. The `fifo` provides the own copy of the obtained data stream for each of its consumers (in Fig. 1 for simplicity only one of them connected through pair of hooks `output1` and `in` is depicted). In the DSS DAQ the `fifo` feeds the ***b2r(1)*** for *ROOT* events production [4] by the full (or possibly downscaled) data stream of trigger events (as well as all Begin–of–Burst (BoB) and End–of–Burst (EoB) events), while the ***b2r(1)*** specially compiled for EoB counts viewing — by all the EoB events only. (Here the Burst means accelerator spill.) The ***ng_mm(4)*** node [4] is used for data transfer over the contexts boundary (kernel to user) as a replacement for the *netgraph* standard ***ng_socket(4)*** node due to more handful bufferization and better performance.

The ***b2r(1)*** directs *ROOT* events to `fifo2` node through the pair of hooks `out` and `input` (see Fig. 1 right part) with mediation of ***ngput(1)*** utility and ***ng_socket(4)*** node. The `fifo2` feeds both the ***r2h(1)*** for histograms filling [4] and ***r2h(1)*** specially compiled to support ***r2h.conf(5)*** objects for polarization calculations (see section 3.1) by the full (or possibly downscaled) stream of *ROOT* representations of trigger events as well as all BoB and EoB events in the *ROOT* representation (in Fig. 1 for simplicity only one of consumers connected through pair of hooks `output1` and `in` is depicted).

The ***b2r(1)*** and ***r2h(1)*** are user context utilities because C++ code and *ROOT* dictionaries and libraries could not be linked into OS kernel. So the data streams cross the context bound-

ary using ***ng_mm(4)*** or ***ng_socket(4)*** nodes and ***ngget(1)*** (for transfer from kernel context to user one) or ***ngput(1)*** (vice versa) utilities.

The *ngdp* generic node types ***ng_fifos(4)***, ***ng_mm(4)*** are explained in [4]. For the *netgraph* generic nodes ***ng_socket(4)***, ***ng_ksocket(4)*** description see the corresponding online manual pages.

The high level control over DAQ graph is performed by special ***ng_vmectl(4)*** control node type intended to issue commands (in the form of *netgraph* control messages) to VME drivers and VME nodes. ***ng_vmectl(4)*** obtains responses to commands as control messages from VME nodes and as data messages from VME drivers, thus supplies the next command from the own queue in time. Instantiation of this node (which is singleton) leads to building of the graph like pictured in Fig. 1.

3 User context utilities

Lets describe only software utilities specific for DSS DAQ system. The *ngdp* generic ones used here are: ***writer(1)*** [17] of packet stream to files on hard disk, ***ngget(1)*** packet stream extractor from *netgraph* graph, ***ngput(1)*** packet stream injector to *netgraph* graph, ***b2r(1)*** (binary–to–*ROOT*) converter, ***r2h(1)*** (*ROOT*–to–histograms) converter, and ***histGUI(1)*** standalone client for ***r2h(1)*** (see [4]). According to the scheme of events representation by *ROOT* classes [4] the specific classes for trigger, BoB, EoB events as well as required helper classes are designed. These classes are linked into ***b2r(1)*** and ***r2h(1)*** as well as provided in the *libElinpol.so* standalone shared library form to be used by the express–offline *ROOT* scripts and offline processing utilities. The *b2r* reads raw data packets and for each of them produces *ROOT* event representation in the form of some compiled-in *ROOT* class. This *ROOT* representation is serialized using `TBufferFile` and encapsulated into a packet of other type with the same number.

The *r2h* reads the data packets with events in the *ROOT* representation produced by ***b2r(1)***, extracts them, fills some histograms configured by <`cfgfile`> in the ***r2h.conf(5)*** format [4], sends requested histogram(s) to registered client(s) by *ROOT* `TMessage`(s), and optionally writes all configured histograms to *ROOT* `TFile` <`outfile`>.

The ***r2h.conf(5)*** protocol [4] allows us to configure the ***r2h(1)*** and ***histGUI(1)*** as well as to establish the conversation between them online. This means we need not to recompile ***r2h(1)*** each time when we calculate other values and book and fill other histograms. Instead we edit the configuration file and restart ***r2h(1)***, which anyway should be restarted at each run if we need to save histograms into per–run files.

The EoB events viewer (so called dumper *b2r_dump*) is a specially compiled form of the ***b2r(1)*** converter, which outputs to terminal instead of the ***syslog(3)*** facility and executed with command line option `-D0x13` to produce textual output instead of *ROOT* events representation.

3.1 Polarization calculator

A specially compiled ***r2h(1)*** converter (so called *r2h_calc*) supports additional entities `Calcvp` and `Calctp` in the configuration file format ***r2h.conf(5)*** [4]. This allows it to calculate polarizations instead of *ROOT* histograms filling. Currently the polarization calculator produces textual output of calculation results at each EoB event arrival. In the future these results could be output in the HTML form to be included into Web–based representation scheme described in [6] and [7].

During the 53rd and 54th Nuclotron runs the LHEP–JINR polarized ions source SPI [15] provided beams with two polarized modes (changing burst-by-burst and called “+” and “–”

below) from the eight ones available for deuterons (or two for protons), as well as the unpolarized beam (so called "0" polarization mode). The corresponding polarization marks are distributed to polarimeters by SPI electronics.

The polarimetry part [8] of the DSS setup allows us to use coincidences of "proton" and "deuteron" scintillation counters in the complementary arms — LD-RP, RD-LP, UD-DP, DD-UP. Lets name these coincidences as Left, Right, Up, and Down arm scaler counts. We have 9 (4) "proton" ("deuteron") counters in Left, Right, and Up arms, and 4 (1) – in Down arm (due to lack of space). So we can kinematically cover the 18°–60° angle range in the lab. system for a wide range of the beam energies. These pairs could be chosen for each beam energy by kinematics and statistics conditions. The LPP–RPP coincidences of the dedicated counters for quasi–elastic pp scattering at 90° in the c.m. are used as monitor counts. For tensor polarimetry we use counts of 4 or 3 coincidence pairs, for vector one — 2. The `Calcvp` vector polarization calculation object uses Left and Right arm scaler counts (of course, at each polarization mode) as well as monitor counts and number of polarized bursts `Nbm` and `Nbp`. All the mentioned arm scaler counts should be either declared in terms of raw data (`Var`) or calculated (`Cvar`) from already known `Var`s by the so called universal calculation mechanism ***cell(3)*** (see [4] for details). This allows us to organize polarization calculations very flexibly without recompilation of the ***r2h(1)*** executable itself. We do not explain too huge details of these counts preparation (at least counts should be normalized to the number of bursts), however, this procedure could be revised easily and quickly. For example, at beam energy change we can operatively replace the scintillation counters combination producing any arm counts, as well as the analyzing power values (which should be already known from some previous measurements). Also the data cuts in terms of 1D–ranges on the time and/or amplitude 1D– and 2D–histograms applied during the counts preparation could be revised. Of course, this is true for additional counts used by the `Calctp` tensor polarization calculation object.

3.2 *sv(1)* supervisor utility

For the human operator convenience the ***sv(1)*** supervisor utility with GUI expected to be self–explanatory is implemented. The main window allows us to perform base DAQ operations like start/stop run, change output filenames, load/unload software components, etc. The VME hardware configuration is represented for human operator by additional windows. To perform base DAQ commands, the supervisor relies on configuration file *linpolsv.mk* in the so called makefile format (see ***make(1)*** for syntax description). This file contains the named targets (i.e. `continue`, `pause`, `init`, etc.) and for each of them it provides some actions in terms of shell commands, while the supervisor simply executes something like `make -f linpolsv.mk continue`. To perform VME hardware configuration, the supervisor uses the ***netgraph(3)*** application program interface (API) to send the *netgraph* control messages to VME nodes. The configuration is represented to human operator by additional windows and is stored in per hardware module files of ***termcap(5)*** format, from which it also could be loaded into GUI. To download the whole hardware module(s) configuration from these file(s) to the VME hardware, a specialized command–string utility ***hwmod_conf(1)*** is implemented, which shares sources with supervisor.

4 Conclusions

• The explained DSS DAQ system was successfully used during the 53rd (December 2016), 54th (March 2017), and 55th (March 2018) Nuclotron runs.
• The A_y, A_{yy}, A_{xx} analyzing powers of the dp elastic scattering and vector and tensor polarizations of the deuteron beam from the SPI [15] have been successfully obtained using the

designed express–offline software in the *ROOT* scripts form.

• For the SPI modes interesting for DSS project the deuteron polarizations for 135 MeV/nucleon beam were calculated by DAQ online in satisfactory agreement with offline data analysis results obtained later [18].

• As SPI methodic studies the polarizations were also measured for other SPI modes [18] with satisfactory agreement between theoretically expected and experimentally obtained values.

• In the future the polarizations could be determined online by the polarization calculator utility using the already known analyzing powers and data cuts in terms of 1D–ranges on the time and amplitude 1D– and 2D–histograms. The polarization calculation results could be integrated into Web–based representation scheme (see [6, 7]). So the DSS DAQ system is suitable for online polarimetry.

• The first measurements of the 500 MeV proton beam polarization were also performed during the 54th run [13].

Acknowledgments. The author has the pleasure to thank V.P.Ladygin for initiation of the polarization calculations development for DSS setup, S.G.Reznikov — for fruitful discussions about VME hardware, and all the DSS collaboration — for permanent support and assistance. The author is grateful to V.V.Fimushkin, A.S.Belov, A.V.Butenko, and all the SPI and Nuclotron teams for cooperation during the runs. The present work could not be done without background provided by LHEP NOAFI team.

The work has been supported in part by the RFBR under grant 16-02-00203a.

References

[1] V.P. Ladygin, Y.V. Gurchin, S.M. Piyadin, A.A. Terekhin, A.Y. Isupov et al., Few Body Syst. **55**, 709 (2014)
[2] A.Y. Isupov, V.A. Krasnov, V.P. Ladygin, S.M. Piyadin, S.G. Reznikov, Nucl. Instr. and Meth. in Phys. Res. A **698**, 127 (2013)
[3] A.Y. Isupov, arXiv:1004.4474 [physics.ins-det] (2010)
[4] A.Y. Isupov, arXiv:1004.4482 [physics.ins-det] (2010)
[5] R. Brun, F. Rademakers, *ROOT – An Object Oriented Data Analysis Framework*, in *Proc. of the AIHENP'96 Workshop* (Lausanne, Switzerland, 1996), Nucl.Instr.and Meth.in Phys.Res. A **389**, 81-86 (1997). See also http://root.cern.ch/
[6] A.Y. Isupov, Czech. J. Phys. Suppl. A **55**, A407 (2005)
[7] A.Y. Isupov, Czech. J. Phys. Suppl. **C56**, C385 (2006)
[8] P.K. Kurilkin, V.P. Ladygin, T. Uesaka et al., Nucl.Instr.and Meth.in Phys.Res. A **642**, 45 (2011)
[9] Y.V. Gurchin et al., Phys.Part.Nucl.Lett. **8**, 950 (2011)
[10] S.M. Piyadin et al., Phys.Part.Nucl.Lett. **8**, 107 (2011)
[11] P.K. Kurilkin et al., Phys.Lett.B **715**, 61 (2012)
[12] V.P. Ladygin et al., JPCS **938**, 012007 (2017)
[13] V.P. Ladygin, Y.V. Gurchin, A.Y. Isupov et al., JPCS **938**, 012008 (2017)
[14] M. Janek, V.P. Ladygin, Y.V. Gurchin, A.Y. Isupov et al., JPCS **938**, 012005 (2017)
[15] A.S. Belov, D.E. Donets, V.V. Fimushkin et al., JPCS **938**, 012017 (2017)
[16] http://afi.jinr.ru/
[17] K.I. Gritsaj, A.Y. Isupov, JINR Communications E10–2001–116 (2001)
[18] Ya.T. Skhomenko, V.P. Ladygin, Y.V. Gurchin, A.Y. Isupov et al., JPCS **938**, 012022 (2017)