Online polarimetry of the Nuclotron internal deuteron and proton beams

A Yu Isupov
Laboratory of High Energy Physics, Joint Institute for Nuclear Research, Dubna 141980, Russia
E-mail: isupov@moonhe.jinr.ru

Dedicated to the bright memory of Prof. L.S.Zolin

Abstract. The spin studies at Nuclotron require fast and precise determination of the deuteron and proton beam polarization. For these purposes new powerful VME–based data acquisition (DAQ) system has been designed for the Deuteron Spin Structure setup placed at the Nuclotron Internal Target Station. The DAQ system is built using the netgraph–based data acquisition and processing framework ngdp. The software dealing with VME hardware is a set of netgraph nodes in form of the loadable kernel modules, so works in the operating system kernel context. The specific for current implementation nodes and user context utilities are described. The online events representation by ROOT classes allows us to generalize code for histograms filling and polarization calculations. The DAQ system was successfully used during 53rd and 54th Nuclotron runs, and their suitability for online polarimetry is demonstrated.

1. Introduction
The precise online polarization measurements of the deuteron and proton beam in the Nuclotron ring are required for many projects of the LHEP research program. The Deuteron Spin Structure (DSS) setup [1], [2], [3], [4] placed at the Nuclotron Internal Target Station (ITS) [5] studies the polarization observables in the \( dp \) elastic scattering for a several years [6]. Vector and tensor polarizations of the deuteron and proton beams from the LHEP–JINR polarized ions source (SPI) [7], 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. These measurements were done using the VME–based data acquisition (DAQ) system which will be described in present paper.

Through the presented text the file and software package names are highlighted as italic text, C and other languages constructions — as typewriter text. Reference to the manual page named “qwerty” in the 9th section is printed as qwerty(9). All 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 — we can see in Fig.1. This picture was automatically generated by the dot command of ngctl(8) utility from graph was really built, and manually simplified by stripping out both nodes non–essential for understanding and details of too technical nature.
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 are hooks named within.

![Diagram](image)

**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 are hooks named within.

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 produced by pairs of so called hooks (octagons in Fig.1). The data messages flow along the edges. For reasons to use the *netgraph* package see [8]. The nodes are implemented in form of the loadable kernel modules (KLD), which approach simplifies and speeds up the debugging essentially. The data processing code execution in the kernel context allows us to handle data obtained from the VME master in the fastest possible way [8].

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 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 an existing quantity of 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 [9]. So the VME driver contains UNIX hardware driver with *probe(), attach(), detach()*, *interrupt handler, kernel thread(s) kthread(9)* to serve a requests *queue(3)* and process data, 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 by mediation of VME driver. Currently we support: the trigger TTCM/FVME2TMWR modules [9] by the *ng_trighwmod(4)* node type, the trigger U40 modules [9] by the *ng_u40hwmod(4)* node type, and the TQDC16 time– and charge–to–digital converter modules [9] by the *ng_tqdcchwmod(4)* node type.

Interconnections in the DAQ graph (see Fig.1) are following: VME driver in $<$N$>$th crate is connected by own ct1 hook to ct1$<$N$>$ hook of *ng_vmectl(4)* node, while VME node in $<$N$>$th slot — by own sl hook to sl$<$N$>$ hook of VME driver of corresponding VME crate.

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

$$\text{ng_mm} \not\rightarrow \text{ng_get} \mid \text{writer}$$

$$\text{ng_melink} \rightarrow \text{ng_fifos} \rightarrow \text{ng_mm} \not\rightarrow \text{ng_get} \mid \text{b2r} \mid \text{ngput} \not\rightarrow \text{ng_socket} \rightarrow$$

$$\text{ng_mm} \not\rightarrow \text{ng_get} \mid \text{b2r} \mid \text{ng_mm} \not\rightarrow \text{ng_get} \mid \text{r2h} \mid \text{histGUI}$$

$$\text{ng_mm} \not\rightarrow \text{ng_get} \mid \text{r2h} \mid \text{histGUI}$$

$$\text{ng_mm} \not\rightarrow \text{ng_get} \mid \text{r2h} \mid \text{polar. calc.} \ldots$$
where:

→ **packet(9)** [10] data stream in the kernel context (along graph edges);

↗ context boundary crossing by the data stream;

1 **packet(9)** [10] data stream in the user context (so called pipe);

→ -- socket connection (here **TServerSocket** / **TSocket** from **ROOT** framework [11]);

... number of the same elements.

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

The **melink0** VME driver node (see Fig.1) is the source of data packets stream which supplied to the **ng_fifos(4)** node [12] named **fifo** through the pair of hooks **data-input**. The **fifo** provides 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-in** is depicted). In the DSS DAQ the **fifo** feeds the **b2r(1)** for **ROOT** events production [12] 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)** for EoB events viewing — by the EoB events only. (Here the Burst means accelerator spill.) The **ng_mm(4)** node [12] is used as replacement for the **netgraph** standard **ng_socket(4)** node due to more handled bufferization and better performance.

The **b2r(1)** directs **ROOT** events to **fifo2** node through the pair of hooks **out-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 [12] and **r2h(1)** for polarization calculations (see section 3.1) by the **ROOT** representation of trigger events (possibly with downsampling) as well as all BoB and EoB events (in Fig.1 for simplicity only one of consumers connected through pair of hooks **output1-in** is depicted).

The **b2r(1)** and **r2h(1)** are user context utilities because C++ code and **ROOT** libraries could not be linked into OS kernel. So data streams cross the context boundary using **ng_mm(4)** or **ng_socket(4)** nodes and **ngget(1)** or **ngput(1)** utilities.

High level control over DAQ graph is performed by special **ng_vmectl(4)** control node type, which 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, so supplies the next command in time.

3. **User context utilities**

Lets describe only software utilities specific for DSS DAQ system. The **ngdp** generic ones used here are: **writer(1)** [10] 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 [12]). According to scheme of events representation by **ROOT** classes [12] 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** shared library form to be used by the express–offline **ROOT** scripts. The EoB events viewer (so called dummy **b2r_dump**) is a specially compiled form of the **b2r(1)** converter.

3.1. Polarization calculator

The specially compiled **r2h(1)** converter supports additional entities **Calcvp** and **Calctp** in the configuration file format **r2h.conf(5)** [12]. 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 [13] and [14].

All used arm scaler counts of the DSS setup polarimetry part (see [2]) are derived from raw data by so called universal calculation mechanism (**cell(9)**, see [12]). This allows us to organize polarization calculations very flexibly without recompilation of the **r2h(1)** executable itself.
not explain too huge details of these counts preparation (at least counts should be normalized to number of bursts), however such 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.

3.2. sv(1) supervisor utility
For the human operator convenience the sv(1) supervisor utility with GUI which expected to be self–explanatory is implemented. To perform base DAQ commands like start/stop run, load software components, etc. the supervisor relies on configuration file linpolsv.mk in the so called makefile format (see make(1) for syntax description). This file contains 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. Also to load the whole hardware module(s) configuration from the file(s) the special command–string utility hwmod_conf(1) is implemented, which shares sources with supervisor.

4. Conclusions
The explained DSS DAQ system was successfully used during 53rd (December 2016) and 54th (March 2017) 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[7] have been successfully obtained using the designed express–offline software in the ROOT scripts form. The first measurements of the 500 MeV proton beam polarization were performed during 54th run also. In the future the polarizations could be determined online by the polarization calculator utility and their results could be integrated into Web–based representation scheme described in [13] and [14]. So the DSS DAQ system is suitable for online polarimetry.

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

References
[1] Ladygin V P, Gurchin Y V, Piyadin S M, Terekhin A A, Isupov A Y et al. 2014 Few Body Syst. 55 709–712
[2] Kurilkin P K, Ladygin V P, Uesaka T et al. 2011 Nucl.Instr. and Meth.in Phys.Res. A 45–51
[3] Gurchin Y V et al. 2011 Phys.Part.Nucl.Lett. 8 950–958
[4] Piyadin S M et al. 2011 Phys.Part.Nucl.Lett. 8 107–113
[5] Isupov A Y, Krasnov V A, Ladygin V P, Piyadin S M and Reznikov S G 2013 Nucl. Instr. and Meth. in Phys. Res. A 127–134
[6] Kurilkin P K et al. 2012 Phys.Lett.B 4 61–65
[7] Finushkin V V et al. 2016 J. Phys. Conf. Ser. 678 012058–012062
[8] Isupov A Y 2010 arXiv:1004.4474 [physics.ins-det]
http://afi.jinr.ru/
[9] Gritsaj K I and Isupov A Y 2001 JINR Communications E10–2001–116 1–19
[10] Brun R and Rademakers F 1996 Proc. of the AIHENP’96 Workshop (Nucl.Instr.and Meth.in Phys.Res. (1997) vol A(389)) (Lausanne, Switzerland) pp 81–86 See also http://root.cern.ch/
[11] Isupov A Y 2010 arXiv:1004.4482 [physics.ins-det]
[12] Isupov A Y 2005 Czech. J. Phys. Suppl. A55 A407–A414
[13] Isupov A Y 2006 Czech. J. Phys. Suppl. C56 C385–C392