Data Acquisition Backbone Core DABC

J Adamczewski, H G Essel[1], **N Kurz and S Linev**
Gesellschaft für Schwerionenforschung GSI
Planckstr. 1, 64291 Darmstadt, Germany

E-mail: H.Essel@gsi.de

Abstract. For the new experiments at FAIR new concepts of data acquisition systems have to be developed like the distribution of self-triggered, time stamped data streams over high performance networks for event building. The Data Acquisition Backbone Core (DABC) is a software package currently under development for FAIR detector tests, readout components test, and data flow investigations. All kinds of data channels (front-end systems) are connected by program plug-ins into functional components of DABC like data input, combiner, scheduler, event builder, analysis and storage components. After detailed simulations real tests of event building over a switched network (InfiniBand clusters with up to 110 nodes) have been performed. With the DABC software more than 900 MByte/s input and output per node can be achieved meeting the most demanding requirements. The software is ready for the implementation of various test beds needed for the final design of data acquisition systems at FAIR. The development of key components is supported by the FutureDAQ project of the European Union (FP6 I3HP JRA1).

1. Introduction

1.1. The New Facilities at GSI

The upcoming new GSI facility FAIR [1] will provide unprecedented accelerator facilities to investigate physics cases in the fields of nuclear structure physics and nuclear astrophysics, hadron physics, physics of nuclear matter, plasma physics, atomic physics, and applied physics.

1.2. Data Acquisition Concepts

One of the most challenging experiments at FAIR is the Compressed Baryonic Matter experiment CBM [2] because it will produce hundreds of GByte/s primary data rate and needs thousands of CPUs on-line for event filtering.

The main concepts of a data acquisition for such an experiment have been presented in [3]. The key idea was not to use traditional trigger mechanisms but rather process time stamped data streams from self triggered front-end components and combine them through a high speed network. All algorithms needed to define events and to filter them out could then run on arbitrary CPUs beyond this building network.

The topology of such a system is relatively straight forward because no trigger decisions have to be distributed. It is not critical to latencies in the order of μsec because the data flows in one direction only. Complex and flexible event selection codes can be implemented more conveniently using

[1] Corresponding author.

commodity hardware. However, two key problems must be solved: firstly the timing system, and secondly the high data rates. Because of the enormous progress in networks nominal bi-directional data rates in the order of GByte/s per node are possible already today. Can such rates be achieved also in data acquisition systems?

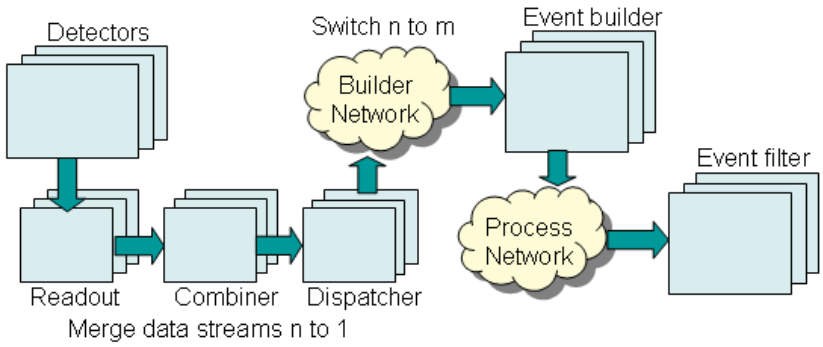

Figure 1. Schematic view of data flow.

The main components of such a data acquisition system are outlined schematically in figure 1. Data signals from the detectors are digitized in readout modules. A time stamp is added (not shown). Special time stamps mark time slices which can be used later for event building. Combiner boards merge the data streams of several readout modules and send them to dispatcher modules.
These modules send data packages through a switched builder network to the event builder nodes (CPU farm). The data streams must contain some information for the dispatchers to be able to determine and identify these packages. This could be special time markers defining time slices as foreseen in CBM, but also bunch crossing numbers, trigger IDs or whatever the front-end systems can deliver. The packages with the same ID from all detectors must arrive at the same event builder. To achieve an optimal network utilization the dispatchers must be synchronized and send their data following a schedule. The schedule is made up and distributed by a master module based on information received from all dispatchers. All scheduler traffic runs over the same builder network. Detailed simulations of such builder networks have been presented in [3].

1.3. Motivation for an Acquisition Backbone

A data acquisition system as sketched above, meeting performance requirements like the ones of CBM, has to be developed in several phases. In the first phase the feasibility of the concept has to be proven, i.e. the data flow performance and the event building. The design of a final data acquisition production system will start after such approval. The concept of the realization of CPU power for the event filters depends strongly on the developments on the market and will be finalized in a later phase.
For the first phase a test bed for FAIR detectors, readout components, data flow investigations (switched event building) and last, but not least, controls is needed. In this first phase the dispatcher boards are built as PCI express cards plugged into standard PCs which perform the event building. The software for that is designed as a data acquisition backbone core (DABC). It is a backbone because it does not include the front-end systems. It is a core which provides interfaces to plug-in custom code for the handling of nearly arbitrary front-end systems.
In the case of a time stamped system a plug-in in the event builders must identify the events inside the data of a time slice. This could be done by multiplicity histograms over time. A peak in such a histogram would identify the time region of an event. The data of that time region from all packages would be composed to the event. In case of a triggered front-end system the events are already tagged. A different plug-in would be needed to compose the events from the tagged data.
Because of the required flexibility the DABC could be also useful as general purpose data acquisition package.

2. Development of DABC

2.1. Hardware Setup Use Case

An example of a hardware setup to be handled by DABC is shown in figure 2. It represents the full data chain as shown in figure 1 at small scale.

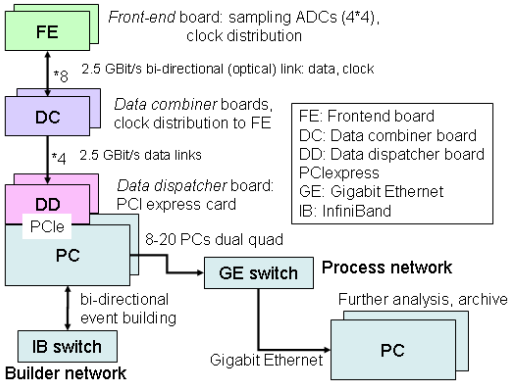

Figure 2. Hardware setup with data chain from front-ends to PCI express cards.

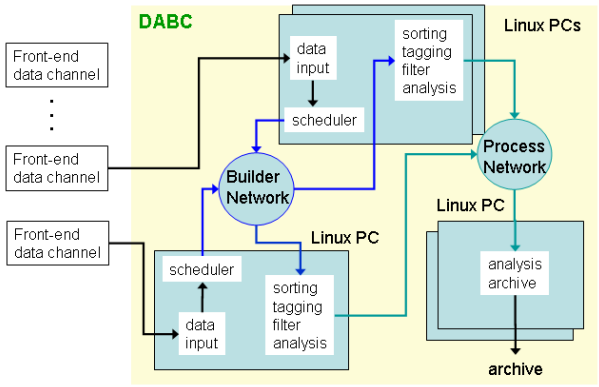

Figure 3. Logical structure of DABC.

Front-end boards typically keep sampling digitizers, whereas combiner and dispatcher boards have fast data links controlled by FPGAs. Prototypes of these boards are under test as they are a first generation of modules needed for the CBM or other experiments. InfiniBand as an event building network has been chosen because of its high performance which is a factor of 10 better than Gigabit Ethernet. DABC must prove that it can really operate with such a high throughput. Other networks like Ethernet are supported as well.

2.2. DABC Design

The topology of software components of DABC are sketched in figure 3 in a coarse grain. Depending on the performance requirements (data rates of the front-end channels) one may connect one or more front-ends to one DABC node. From a minimum of one PC with Ethernet up to medium sized clusters all kinds of configuration are possible. One may separate input and processing nodes or combine both tasks on each machine using bi-directional links depending on CPU requirements.

The data of one event or time slice from the input channels of each PC, e.g. Ethernet or PCI modules, is sent by a *scheduler* over the *Builder Network* to one receiver node where it is composed, sorted, tagged etc. Receivers then may send composed data over the *Process Network* to further analysis or archive nodes. Using a separate network improves the data throughput.

2.3. Implementation

As mentioned above, the DABC should not be very specific to front-end systems. As a consequence it must provide convenient mechanisms to integrate application specific code to handle front-end hardware, data formats, etc. DABC is written in C++. Application code is plugged in as implementations of interfaces defined by DABC or subclasses of DABC classes.

The basic component is a *Module* which gets access to *Data queue* buffers through input *ports*, processes them, and outputs them to output *ports* as shown in figure 4. Because a *Module* can have several input *ports,* it can also merge more than one input data streams into a new output buffer. A central *Module manager* keeps all *Modules* of the local application and provides a global interface for module set up and communication. The modules and their components (e.g. ports, parameter objects, command descriptors) are organized by name in a hierarchical folder structure.

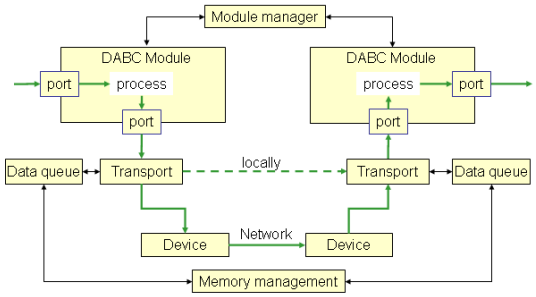

Figure 4. Module communication through ports.

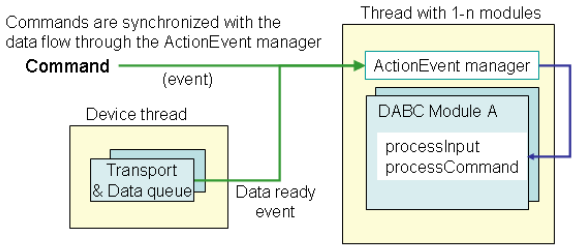

Figure 5. Thread synchronization and module control.

The *Data queue* buffers are controlled by a central *Memory management* consisting in a set of memory pools. If modules run on the same node, this allows to transfer buffers between them "locally" without copying their contents. Here the *Transport* component just propagates buffer references. If a *port* is connected to a remote *port*, the *Transport* and *Device* components do the transfers over a network. A *Device*, e.g. a socket device, can control several *Transports*. *Transport* and *Device* components are implemented for TCP and InfiniBand *Verbs* (see section 3.1). Others could be added, e.g. Myrinet.
Module components can either run in separate threads to utilize multiple CPUs, or in one thread which is faster because no module synchronization by mutex is needed in this case. Instead of locking all critical sections, the data processing functions of the modules are synchronized within one thread by an *Action event manager* as shown in figure 5.
The data packets are contained in buffer resources from the *Memory Management* (memory pool). The references to these buffers are propagated between module ports by means of *Data queues* that belong to the connecting *Transport* object (see also figure 4). If the *Transport* thread adds an arriving queue buffer to a module input *port*, this *port* will implicitly send a *"Data ready event"* signal to the *ActionEvent manager*. The manager will react on the event by calling the *processInput* function of the module associated with the signalling port. This function will get the buffer reference from the port. It then may work on the buffer data, forward it to a module output port, or eventually give the buffer back to the memory pool ("free" the reference).
Similarly, if a DABC *Command* object arrives at the command input queue of a module, a different signal will be sent to the *ActionEvent manager*. The manager thread then calls the *processCommand* function of the associated module which will perform the command action.

2.4. Data Flow Engine

As shown by the InfiniBand performance evaluations (see section 3) this network is very powerful in optimizing non-synchronized data traffic. However, in these measurements the receiver nodes always had enough queue buffers. If the receiver nodes are busy with other calculations like event building, it is necessary to hold the senders to avoid resource exhaustion at the receiver side. This is achieved by a back pressure mechanism [4].

3. Measurements

Before we started with the design of DABC very detailed measurements and tests of hardware and software components had been performed.

3.1. InfiniBand Hardware and Software Setup

We evaluated InfiniBand (IB) as an excellent candidate for event building over network in a first prototype of DABC. InfiniBand provides high data rates with low CPU utilization (few %), guaranteed delivery, and low latency (few µsec). A small test cluster of four nodes (dual Opteron) was installed at GSI in November 2005. Each node of that cluster is equipped with a Mellanox MHES18-XT

InfiniBand Host Channel Adapter (HCA). The nominal data rate of such adapters is 10 Gbit/s in full duplex mode.
All software required to configure and operate InfiniBand networks is collected in the OpenFabric Enterprise Distribution OFED (version used: 1.2), developed by the OpenFabrics Alliance [5]. Several user-level programming interfaces (API) were investigated and tested.
The low level InfiniBand *Verbs* programming interface included in the OFED package provides direct access to the HCA functionality from user space (so-called kernel bypass). Its most important functionalities are non-blocking zero-copy data transfer, remote direct memory access (RDMA) and (unreliable) hardware multicast.
The *User Direct Access Programming Library* (uDAPL), developed by the DAT collaborative [6] was inspired by IB functionality. Therefore it has many similarities with *Verbs* API. Since uDAPL uses a peer-to-peer communication paradigm, multicast is not supported.
The *Message Passing Interface* (MPI) is widely used in the field of parallel computing. It defines an API for fast exchange of data between computing nodes. The MPI over InfiniBand project MVAPICH [7] provides non-blocking zero-copy data transfer and hardware IB multicast.

3.2. InfiniBand Benchmarking

A test application was written to evaluate InfiniBand performance with all mentioned APIs. This test application is capable of generating different kinds of traffic patterns over InfiniBand. Two modes are possible: synchronized and non synchronized. In synchronized mode the clocks of the nodes are synchronized. The senders perform time scheduled data transfers. This avoids congestions at the receiver nodes. Figure 6 shows results of an all-to-all traffic pattern, where each node transfers data to all other nodes including itself according to a round-robin schedule with synchronization. The dependency of achieved data rates per node on the package size for the three APIs is presented.
All APIs provide good performance and reach 900 MByte/s for packet sizes greater 64K. Since *Verbs* has less API overhead, it reaches such data rate already at 8K package size and a further 950 MByte/s with 16K packages.

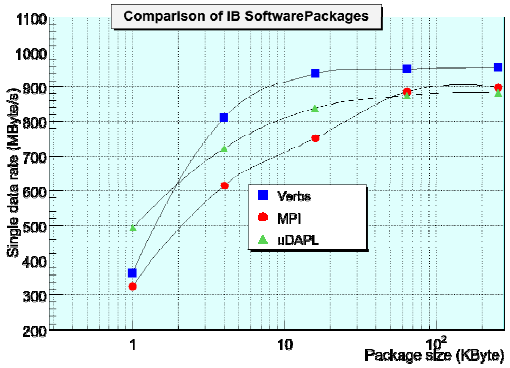

Figure 6. Single node rate performance comparison between different InfiniBand APIs (4 to 4 nodes).

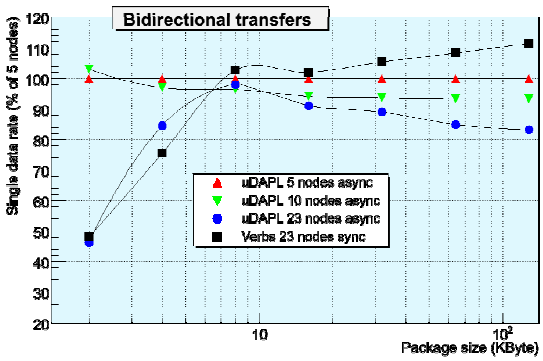

Figure 7. Network utilization with different number of nodes.

3.3. Scaling on Larger Clusters

To extend the measurements to larger clusters cooperations with the Forschungszentrum Karlsruhe [8] and the University of Mainz [9] have been established, respectively. In Karlsruhe, Twenty-three double dual-core Opteron machines were available connected by an InfiniBand SilverStorm (QLogic) switch. Figure 7 shows the scaling of all-to-all transfers from 5 to 10 and 23 nodes, relative to 5 nodes non synchronized (100%). Remarkably the throughput with 6-10 KB buffers scales with nearly no loss indicating the InfiniBand network is able to optimize such traffic. The throughput falls to 84% at large package sizes indicating non-resolvable collisions. For this case synchronization results in 112%.

The effect of scheduling (synchronizing the senders) is shown as squares in figure 7 ("Verbs 23 nodes sync"). With synchronization the throughput can be enhanced for large buffers because now collisions are avoided. Using *Verbs* the synchronized throughput on 23 nodes is even slightly larger than the non synchronized throughput with uDAPL on 5 nodes. On the four nodes at GSI we see no effect of synchronization. One can expect that the benefit of scheduling increases with the number of nodes. Up to 24 nodes can be connected through one single switch chip.

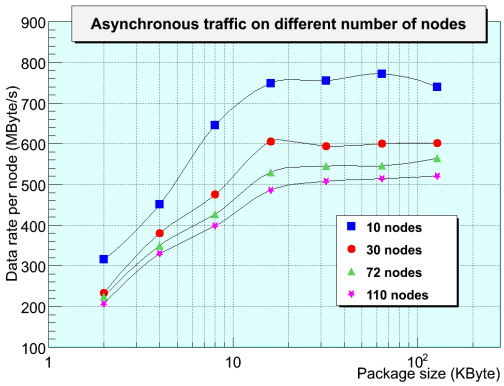

Figure 8. Asynchronous traffic for different number of nodes.

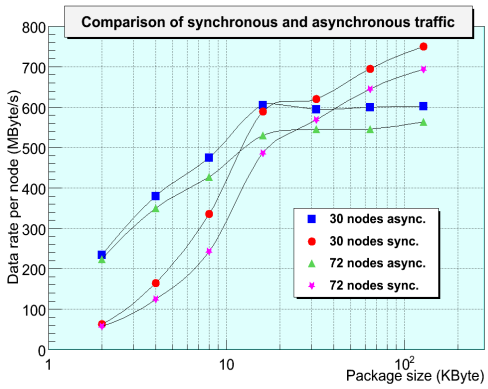

Figure 9. Performance improvements by synchronization.

Therefore further tests have been performed at the *Zentrum für Datenverarbeitung der Johannes Gutenberg Universität, Mainz.* These measurements used up to 110 dual core Opteron nodes equipped with Mellanox MHGS18X InfiniBand host channel adapters, connected by an InfiniBand Flextronics F-X430080 switch (max 144 Ports DDR). Figure 8 shows the data rate versus the package size of all to all traffic for increasing numbers of nodes at the Mainz cluster using *Verbs* in non-synchronized mode. Note that performance decreases with node number over the whole range of package sizes. In figure 9 the effect of synchronization with respect to the network topology is plotted, as measured in Mainz (*Verbs* API).
In case of 30 nodes, each of 5 nodes connect to one switch module (5x6 topology); for 72 nodes, each of 8 nodes connect to one switch module (8x9 topology). This topology is taken into account when scheduling the packets in a synchronized way. The increase of data rates for large packages is similar as the one shown in figure 7. The scheduling strategy of the test does not work well for small packages, because here the IB data rates vary stronger and the scheduler adjusts the timing to the slowest transfers.

3.4. DABC Event Building Performance

A data flow chain with four data generators as data input per node, one combiner per node and one event builder per node has been implemented using the DABC core and plug-in mechanism. The result for InfiniBand and Gigabit Ethernet (GE) is shown in figure 10 in comparison to the test program (*Verbs* test, same as in figure 6). With GE and TCP 70 MByte/s can be achieved, whereas with InfiniBand over 900 MByte/s. The CPU utilization is only 50% (dual CPU). This result approves that DABC does not slow down the data throughput as derived with plain test programs. However, it takes more CPU time, and the package size must be above 32 KByte. The CPU utilisation per MByte/s for InfiniBand is an order of magnitude smaller then the one of TCP with GE because it uses zero-copy DMA. The TCP stack copies the data before sending. With 10 GE and standard TCP the CPU would slow down the theoretical data rate. One has to use other upcoming networking protocols providing zero-copy DMA over 10 GE to get similar results as with InfiniBand. Such protocols could be easily implemented in DABC. Currently we do not have access to 10 GE clusters.

The buffer size is not a critical parameter because one can always combine or split the data of a time slice in the schedule table. DABC supports the InfiniBand gather DMA mechanism avoiding the need to copy data to combine.

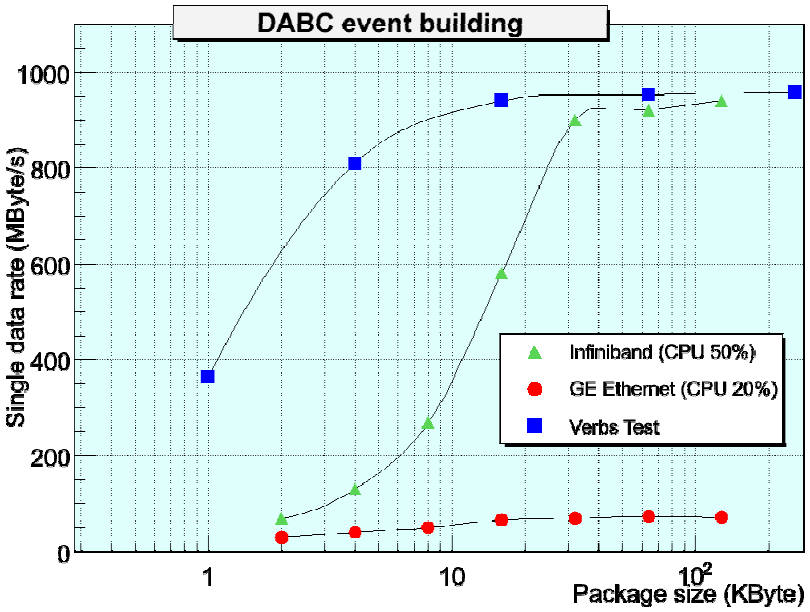

Figure 10. Event building with four nodes.

4. Conclusion

The first version of DABC is able to build events over switched networks at very high data rates as required for an event building network in future FAIR experiments. It provides a test bed for proving various technologies. The choice of the network and its protocol APIs depends on the development of the market. There is no clear preference by performance or functionality.

The software provides interfaces to plug-in custom code for custom data sources. The next task is to build up a full data flow chain from detectors over readout boards, combiner boards, dispatcher boards into DABC. These boards are under construction. Only with this full data chain the goal of phase one can be achieved.

First tests of performance scaling up to 110 nodes show encouraging results, but no dramatically breakdown. To improve performance the data transfers should be synchronized. This can be done within DABC if real-time features of Linux are activated. Besides InfiniBand other fast networks like 10 Gigabit Ethernet can be supported, but up to now no test beds are available.

For controls, DABC can use infrastructure components of the existing XDAQ data acquisition package [10], e.g. an application runtime environment with state machines, info space process variables, and HTTP/SOAP web services. However, the DABC core functionality does not depend on XDAQ classes. Additionally, DABC provides the publication of global parameters to DIM (Distributed Information Management) [11] servers. DABC also defines and executes DIM commands. Currently a simple generic Java based graphical user interface is available.

Because of its flexibility the DABC will replace the event builder of the GSI standard data acquisition system MBS [12]. It can also be used as general purpose base for the implementation of dedicated data acquisition systems in the next years. The DABC web site is available at [13].

5. Acknowledgements
We acknowledge the support of the European Community-Research Infrastructure Activity under the FP6 "Structuring the European Research Area" programme (HadronPhysics, contract number RII3-CT-2004-506078).
For providing resources and support for the large-scale measurements, we thank Frank Schmitz, Ivan Kondov and Project CampusGrid [8] at the Institute for Scientific Computing, Forschungszentrum Karlsruhe; as well as Klaus Merle and Markus Tacke at the Zentrum für Datenverarbeitung der Johannes Gutenberg Universität, Mainz [9].

6. References
[1] H.H. Gutbrod (Editor in Chief), FAIR Baseline Technical Report, ISBN 3-9811298-0-6, EAN 978-3-9811298-0-9
[2] CBM technical status report, GSI, January 2005, pp.235
[3] H.G.Essel, FutureDAQ for CBM: On-line event selection, *IEEE TNS* Vol.**53**, No.**3**, June 2006, pp 677-681
[4] H.G.Essel, Data Acquisition Backbone Core DABC, *IEEE TNS* Vol.**55**, No.**1**, February 2008, pp 251-255
[5] OpenFabric Alliance OFED, - http://www.openfabrics.org
[6] uDAPL, - http://www.datcollaborative.org
[7] MVAPITCH, - http://nowlab.cse.ohio-state.edu/projects/mpi-iba
[8] Project CampusGrid, - http://www.campusgrid.de
[9] Zentrum für Datenverarbeitung der Johannes Gutenberg Universität Mainz, - http://www.zdv.uni-mainz.de
[10] J. Gutleber and L. Orsini, XDAQ framework, - http://xdaqwiki.cern.ch/
[11] C. Gaspar, Distribution Information Management system DIM, - http://dim.web.cern.ch/dim
[12] H.G. Essel et al., The general purpose data acquisition system MBS, *IEEE TNS* Vol.**47**, No.**2**, April 2000, pp 337-339
[13] DABC web site, - http://www-linux.gsi.de/~dabc/dabc/DABC.php