E852 Software note

December 8, 1992

Data Acquisition for E852.

This note has been written by *N.Sinev* on the basis of information provided by *A.Olchanski, A.Ostrovidov and D.Weygand,* and gathered from all available E852 manuals.

Table of contents:

1.E852 hardware structure.
2.Data acquisition organization.
  2.1 DAQ hardware.
  2.2 DAQ processes.
  2.3 Processes timing.
  2.4 Event types.
3.DAQ control.
  3.1 DAQ configuration
    3.1.1 What is resources?
    3.1.2 Where resources are stored?
    3.1.3 Resources names structure.
    3.1.4 Tight and loose binding.
    3.1.5 How to configurate DAQ.
  3.2 DAQ menu
4.Libraries to access hardware through DAQ.
  4.1 Module addressing.
  4.2 Interaction with main DAQ process.
  4.3 Subroutines to perform particular actions.
  4.4 Error codes.
  4.5 Where to find libraries?
5.On-line data access.
  5.1 Dispatcher operation.
  5.2 Library of routines to access data through dispatcher.
6.Devices data formats and timing.
7.DAQ data formats.
  7.1 DAQ raw data formats (before event builder).
  7.2 Tape data format.
  7.3 Dispatcher data format.
8.Device mapping.
9.DAQ speed.
10.Data rates

This is very preliminary mock-up of the note, which I intend to compile with the help of the people involved in DAQ system creation. I ask everybody to send any coments on this note to **sinev@bnlcl6.bnl.gov** *or* **bnlcl6::sinev.**

1. E852 hardware structure.

The E852 setup can be considered as combination of the next subsystems:

1.Beam tracking and monitoring,
2.Soft photon detector (CsI barrel),
3.Soft charge particle detector (cylindrical drift chamber, TCYL),
4.Downstream photon veto endcap array (DEA),
5.Downstream charge particle tracking system(TPX1,2,3, MPS drift chambers and TDX4),
6.Photon spectrometer (LGD),
7.Triggering scintillator counters (S1,S2,S3,CPVA,CPVC,BV,EV).

Each subsystem consists of detectors themselves,their power supplies, and readout and control electronics. From the point of view of the data acquisition system (DAQ), only readout electronics is essential. However, the DAQ system is intended not only to collect data, but serve as interface between user and any computer-controlled devices.

Table 1 shows the detector readout and control electronics. The overall view of E852 electronics system is shown in Figure 1.

2.Data acquisition organization.

2.1 DAQ hardware.

The DAQ system is based on the SLAC Scanner Processors (SSP's), residing in FastBus. They control the data flow and allow maximum data aquisition speed. The current assumption is that the SSP's contain sufficent internal memory to store all data, received during the spill however, external memory is being considered. After the end of spill data are transferred from the SSP's to the host computer, a SGI-420.

The link between FastBus and the host computer is through a BNL custom device, the BranchBus-FastBus controller (BBFC), designed by C.Zein of OLDF, and Omnibyte VMEbus-BranchBus controller (VBBC). BBFC resides in one of FastBus crates, and is connected via BranchBus cable to the VBBC, residing in one of the SGI- 420 VMEbus slots.

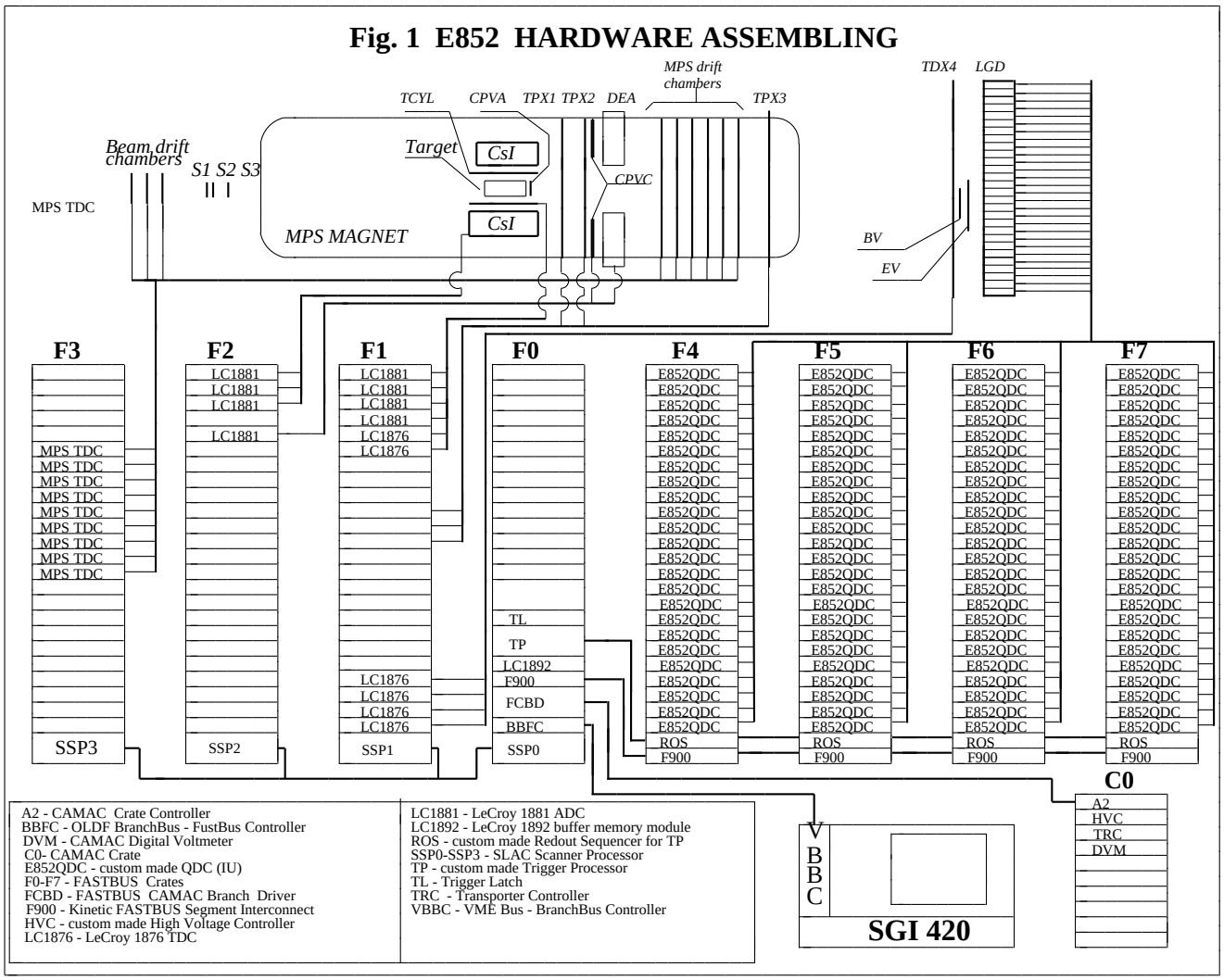
Due to design limitation of the BBFC the host computer can address only the crate containing the BBFC. The use of the SSP's overcomes this difficulty. SSP0, residing in

Table 1. Detectors readout.and control electronics

Detector	Channels	Redaout electronics	F/C	Modules
Beam chambers		MPS made modules	FB	2
CsI	198 ADC	LeCroy 1881 ADC	FB	3
TCYL	128 TDC	LeCroy 1876 TDC	FB	2
	256 ADC	LeCroy 1881 ADC	FB	4
DEA,CPVA,CPVC,	14 ADC	Le Croy 1881 ADC	FB	1
TPX 1,2,3	2016 wires	?	FB	
MPS drift chambers	~10000	MPS made modules	FB	7
TDX4	384 TDC	LeCroy 1876 TDC	FB	4
LGD	3053 ADC	IU made QDC	FB	96
LGD transporter	2	?	C	1
LGD high voltage	3053	MSU/IU made controller	C	1
Trigger Processor		IU made module		
Trigger logic MLU	8	LeCroy 4508 MLU	C	1
Trigger Latch		?	FB	1
Programmable Delay	8	LeCroy 4518	C	1
Timing History	8	LeCroy 1876 TDC	FB	1
Scalers	8	Struck STR200	?	1
Magnets currents monitoring	23	?	C	2

the same crate as the BBFC, acts as a mediator for communication to the other crates. SSP0 communicate with other SSP,s through a standard FastBus cable segment connecting all SSP's in the system. SSP0 communicates with the LGD crates (which do not contain SSP's) through SEGMENT interconnects. (KineticSystem F900) During initialization speed is not primary concern. Initialization operation are carried out by a software system developed by R.Hackenburg,and modified by A.Ostrovidov and discussed in a separate note. During data acquisition high speed operation is essential. SSP0 serves as a buffer for data transferred from low level SSP's. SSP0, without intervension of the host computer, uses block transfer to read data, stored by each low level SSP. The speed of this transfer is limited by the cable segment and is 10 Mbytes/second. SSP0 then passes the collected data to the SGI-420 via BranchBus interface, described above. This transfer speed is 15 Mbytes/sec. Data rate limits, derived from these numbers are discussed in the section 10.

Another factor, which could effect DAQ speed is the size of SSP internal memory. This limit, currently it is 512 kbytes is reduced by 10-20 kbytes for program storage space required for SSP operation. This limit gives an event rate of more than 2000 events/spill if the avarage size of an LGD event is 200 bytes (the current lowest



estimate). If, however, the average LGD event is 1200 bytes (the current highest estimate) the event rate drops to 400 per spill.

There are two ways to avoid this limit, expand the SSP internal memory or use external memory in the form of LRS 1892 FastBus memory module. The first option is most attractive since it does not effect the transfer rate from the LGD QDC's to event storage buffer.

If the LRS 1892 module is used, the readout time for the LGD doubles. The data are first read into the SSP, and then are copied from the SSP to the 1892. This time increase may not be our limiting factor because the event acquisition time is determined the

slowest device,in this case the MPS TDC's.

2.2 DAQ processes

The DAQ system consists of processes, running on the DAQ host (the SGI-420) and on other computers.The relationship of these processes is illustrated by figure 2.

1a) DAQmenu can start the main DAQ process running on the host SGI-420. This program can, in turn, load the SSP's with the code required for data collection, initialize the hardware and execute application specific programs. Examples of application specific activities are setting the PMT high voltages, reading or setting discriminator thresholds, reading or adjusting pedestals, downloading or reading the trigger MLU's etc.

2) The interaction of these processes with the DAQ hardware and the time sequence of activities is as follows:

2a) During the spill triggers are sent to SSP0 (through a special trigger input) causing SSP0 to instruct the lower level SSp's to execute a data acquisition cycle. Since each low level SSP has access to only a subset of the apparatus readout electronics, the data from the event causing the trigger is distributed over several SSP storage buffers. Events are accumulated sequentially for the duration of the spill. 2b) The DAQ main process polls the hardware at some unspecified low rate to determine if an end of spill has occurred. Upon detection of this event, it initiates a readout of the buffered events through SSP0. SSP0 reads the low level SSP's and sends the data into the host SGI shared memory. After readout the SSP's are ready to acquire the next spill of data.

2c) The event builder process, running on the SGI-420, accesses the shared memory and assembles the '' event fragments'' into the form they will be recorded on tape.

2d) One or more tape writing processes access the reassembled events and send them to 8mm tape. These tape drives have a writing speed of 500 kbytes/second which is to be compared to 1-1.5 Mbytes/second of acquired data. This implies that 2-3 tape drives will be running simultaneously. The smallest unit of data to be written to tape is the spill so that on a given tape all the events will be in sequence but not nesseccarily contiguous.

2e) The ''dispatcher feeder'' will access the reassembled events residing in shared memory and pass them to the dispatcher. Every event will be passed from the feeder to the dispatcher. Both of these processes execute on the SGII-420. The dispatcher sends these events, via the network, to any waiting analysis processes. The dispatcher can also distribute the results of these analyses to other processes, e.g. the display programs. The processes ''connected'' to the dispatcher need not be executing on the DAQ host.

There may be other processes running on the DAQ host. For example, the MPS group has requested access to the shared memory containing the event fragments to produce events dumps for the purpose of debugging the drift chambers. This method of accessing the data is faster and more direct than the methods outlined above but cannot be extended to a multi-computer environment.

2.3 Process timing.

Processes, running on the SSPs are tightly synchronized with accelerator cycles. This synchronization is provided by SSP0, which uses the trigger signal, formed by TRIGGER PROCESSOR, as restart signal, to restart data recording. The SSP0 then issues signals to all slave SSPs, and wait until everybody report completion of their tasks.After that SSP0 issues signal to TRIGGER PROCESSOR about readiness to acquire next event. The time, which the event recording takes, depend on many factors, and will be discussed in details in section 9.

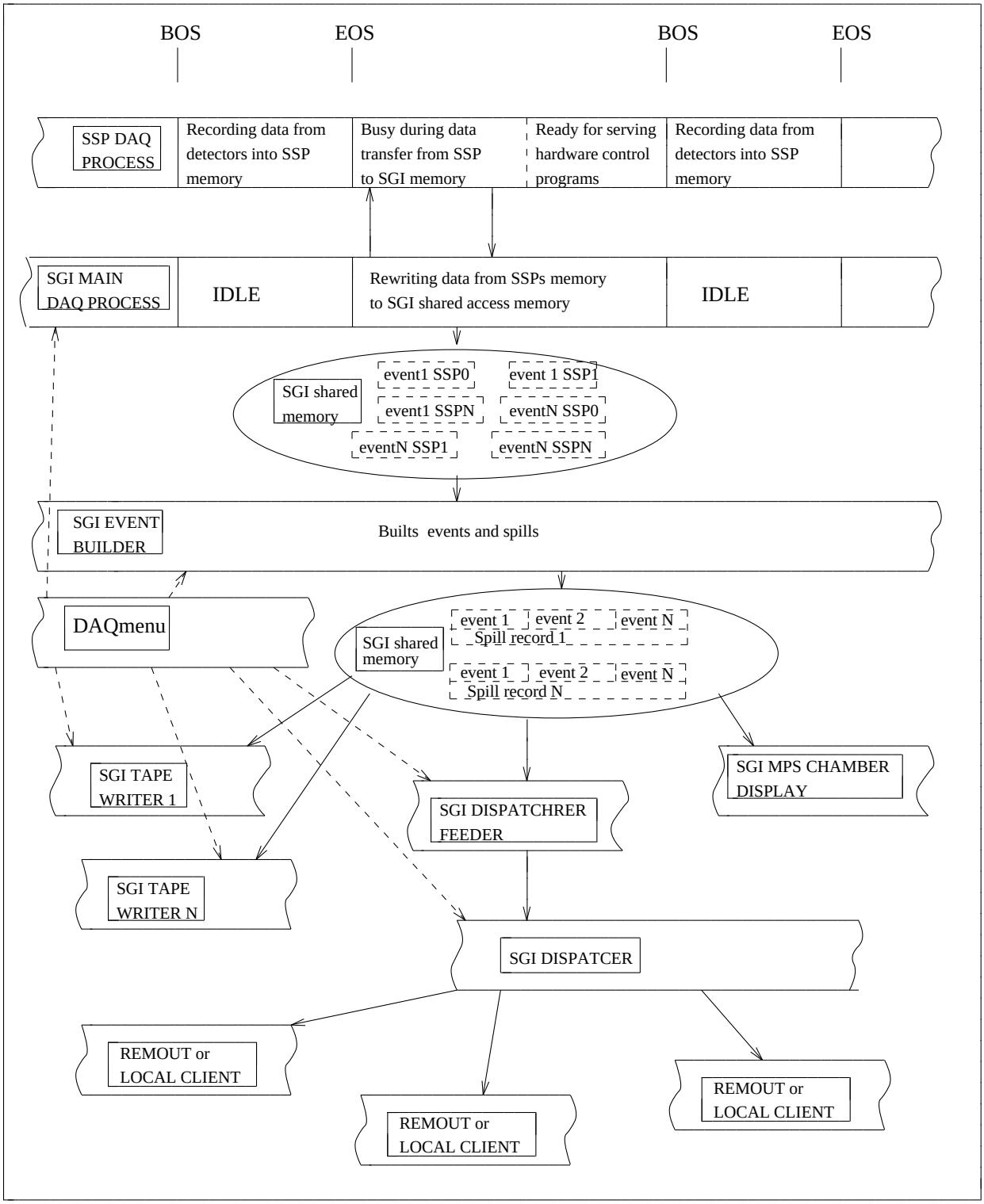
While SSPs are busy with spill data acquisition, the main DAQ process on host SGI420 is idle. It’s activity starts only after SSP0 receives END OF SPILL event (see below about event types).In the interspill time main DAQ process copies all data from SSPs into SGI memory. After completion of data transfer, main DAQ process checks, if there is PAUSE request from DAQmenu (such request may be generated whether by operator, or by any other process, which needs to obtain hardware access (for example, to change transporter position). If there is no PAUSE request, main DAQ process allows SSP0 to assert readiness for next spill data acquisition.

All other DAQ related processes are not directly synchronized with spills. Their activity is determined by the presence of data in corresponding buffer.

2.4 Event types.

The DAQ system can acquire several different types of "event", in addition to the data event, for example, pedestal events, monitor events, beginning of spill, end of spill, etc. Te DAQ system distinguishes different types of events by the values of bits in the trigger latch. The trigger latch is set by the hardware components that formed a particular trigger. The behavior of the system depends on the trigger type an can be modified by changing the configuration of "resources"(see section 3.1). For example, it is possible to configure the system to read out only part of the hardware for a given type of trigger. Currently, the system can accomodate 8 different trigger types. Currently anticipated are end of spill (EOS), beginning of spill (BOS), physics trigger and up to 5 other types of trigger.

Fig.2 DAQ processes



3. DAQ control.

3.1 DAQ configuration.

There are no specific hardware requirements for this system other than the VBBC-BBFC interface from the host SGI-420 and the SSP's.Other parameters of hardware configuration,such as the number of crates, the modules in each crate, the readout procedure for each type of trigger and so on can be changed without recompiling of the system. This flexibility is achieved through the usage of Xlib resource manager subroutines. The general rules for Xlib resources specification can be found in Xlib programing manual in the section devoted to resource manager. Detailed understanding of these specifications should not be needed to operate the DAQ system since an interactive procedure for changing these resources is provided by the DAQ system. An overview of the concepts are given below.

3.1.1 What is resources?

) .ti+3n Resources are similar to FORTRAN parameters. The values, given to these resources determine the actions taken by the program under a given set of circumstances. The advantages of this method compared to parameters are:
1) a resource can be changed without re-compiling the program

2) there is a well defined method to evaluate the resource default value.

Recompilation is required only if the number of SSP's being used exceeds 11, a very unlikely event. Some significant changes in the hardware configuration (adding a new TYPE of FASTBUS module, for example) require that the code loaded into the SSP's be recompiled but the system performs this task automatically at initialization time so this process is transparent to the user.

3.1.2 Where resources are stored?

While DAQ is running the value of resources are stored in the intrinsic RESOURCES DATABASE. It is possible to save this database to a file, however, another mechanism for preserving this information exists. The values of the resources are calculated at initialization time from information contained in the RESOURCE SPECIFICATION file, which may be saved as a way to record the DAQ configuration.

The resource specification file is an ASCII file and may be changed with a text editor. It need not specify every resource value. The resource values are calculated by the resource manager using matching rules and wildcards.

3.1.3 Resources names structure.

Resource names reflect the hierarchical structure of the objects controlled by them. The DAQ system deals with crates, each crate contains modules, each module may perform several functions and so on. Reflecting this structure is the form of a resource

name:

daq.cratename.modulename.operationtype...

The elements of the name seperated by "." are referred to as components or subjects.

Another important concept of the resource manager is the class name. Every subject (or component) of a resource name belongs to a class of subject. For example, cratename may be the name of particular fastbus crate, to be specific, SSP number 3 (in this syntax, ssp_0x3). However, all SSP controlled crates belong to one class of subjects, specifically (in the resource database) Ssp. When a program needs to determine a resource value form the database it passes the resource name (the "instance" name) and the class name to the resource manager. If the resource specification file contains no explicit specification for the requested resource, the resource manager searches for a matching class specification and assigns this value to the resource.

A rule for assigning class names and instance names is that class names USUALLY begin with an upper case letter, and instance names begin with lower case letter. This rule is not strictly obeyed in the current resources database. For example, there is class name for all 1881 ADC which is simply 1881, and instance name for the 1881 ADC in particular position, 1881_0x{hex-address}.

3.1.4. Tight and Loose Binding.

When creating a resource specification file it is possible to use wildcards for the components of resource names. Components may be concatenated with "." or "*". If "." is used (tight binding) the specification defines only the resource matched exactly by the name. However, if "*" is used between components (loose binding) the component(s) occupied by the "*" need not be matched by a resource name to give that resource a value. For example, if there is a string in the resource specification file:

daq*use1892: yes

then, unless explicitly specified otherwise, the program will consider the value of a resource named daq.ssp0x{n}.use1892 defined as yes AND the value of a resource named daq.ssp0{n}.1892_0x{m}.use1892 defined as yes.

3.1.5 How to configure the DAQ.

An example configuration file, default.db, exists. It contains detailed comments and should be understandable as an example the correspondence between resource specifications and the particular parameters of the DAQ system. This file should not be edited. There is also a default working configuration file, e852.db, that may be edited to change the DAQ configuration. It is also possible to change the configuration

interactively at initialization time. The changed file may then be saved.

3.2 DAQ menu.

Initialization and configuration of DAQ processes are only two examples of DAQ processes that can be controlled by the user. Additionally the user has the ability to start and stop data taking (runs), control tape drives, temporarily suspend data taking (pause), redirect the output to disk, etc. These utilities are provided by the DAQmenu process, running simulataneously with the DAQ main process. DAQmenu supervises the performance of all DAQ related processes via an interprocess communication facility. A graphical, menu driven interface is provided for this facility.

4. Libraries to access hardware through DAQ.

Main DAQ process takes care about gathering data during the run and recording it to the tapes. On-line data analysis is made by the set of users programs, which communicate with DAQ through DISPATCHER. Another set of programs should be created to deal with hardware control,testing and tuning. These tasks are not performed by main DAQ process, but code, residing in the SSPs, and subroutine libraries, developed by R.Hackenburg at BNL and partially modified by A.Ostrovidov provide the tool for everybody to create his own code to deal with E852 hardware.

4.1 Module addressing.

As it is seen from the fig.1, there are four different paths for host computer to reach modules in FASTBUS or CAMAC crates.

- Modules, residing in "master" fastbus crate (F0 on the picture), i.e. the crate, where BBFC board resides, are reachable directly through BBFC-VBBC connection. We will refer to such path as DIRECT path.None SSP is involved in that case, so this path does not require any code be loaded into SSP memory, and it has no restriction for the time of access - modules in the master crate could be accessed even while spill acquisition is in progress.
- Modules, residing in the fastbus crates, which are connected to the master crate through SSPs interconnection (F1,F2,F3 on the picture), are accessible through so-called SSP path. This path involves SSP0 and one of the "slave" SSP, so the appropriate code should be loaded to the SSPs before usage of this path. Usually this code is loaded by main DAQ process, or, if DAQ process is not running, DAQinit program should be executed before one runs its own program.This way has also restriction on the time of access. While spill acquisition is in progress, or spill data transfer to host computer is not completed, programs are not allowed to access

modules through SSP path, as it may cause data damage or SSP malfunction.

- Modules in LGD fastbus crates are accessible through segment interconnects (SI), one of which is residing in master fastbus crate, and others - in LGD crates. The path to reach such modules we call SI path. If BBFC have not such drawback, as 5 bit primary address field, this way would not involve SSP in the operation. However, due to 5 bit limitation it is necessary to use SSP0 as mediator in such operation.
- Modules in CAMAC crate are reachable through FBD path. We are going to keep FBD in the master fastbus crate, however, in general, it may be installed in any other crate. Argument fbd-pa in function calls for operations with CAMAC modules depends on the position of FBD board. If FBD board resides in the master fastbus crate, FBD path does not involve any SSP in operation and so free from limitations, affecting SSP path.

Subroutines to access modules through different paths are the same (except FBD path), though particular actions of subroutine depend on the path. To provide subroutines with easy way to determine the path to be used in the particular case, the module address PA must be formed according to the rules, described in the table 2.

Table 2. Rules for primer address (PA) calculation.

Method of module access	Byte 3	Byte 2	Byte 1	Byte 0
Direct	0	0	0	N1
SSP path	0	N3	N2	N1
SI path	N2	X	X	N1
CAMAC crate	FBD_PA			

N1- module slot number
N2- cable address of SSP or SI, serving crate, in which module is.
N3- SSP0 slot number
X - any number (ignored)
FBD_PA - primary address of FBD module, formed according to above rules

There are also some other restrictions and rules for addressing:

- *SSP in master crate should not be placed in slot 0.*
- *SSPs and SIs must have cable addresses in the range 1-31.*
- *SSP path PA with N3=0 will not cause an error, but is used to access memory of slave SSP. Users should carefully avoid such PA values to not destroy the contents of SSP memory.*
- *In the case of a few SIs in the master crate, each with separate cable segment subtree, library provides possibility to serve up to 4 such subtrees, with possibility to have the same values of cable addresses in different subtrees. In that case SIs in master crate will be enumerated and SI with lowest slot number in master crate will*

be #0, next SI - #1 and so on. To access module in one of such subtree it is necessary to place this SI number into bits 30,29 of PA, while cable address of slave SSP (N2 in table 3, SI path) will occupy bits 28-24.Bit 31 of PA is used to access internal SI registers and should be 0 in users programs, with one exceptions, described below.

- *In the case of broadcast addressing, for crates on the SSP path only local broadcast is supported. For direct path and SI path, local and global broadcast are implemented, following standard broadcast addressing rules.Special provision has been made to provide ability of broadcast operation only on selected subtree in the case of a few SIs residing in master crate with separate cable subtree connected to each SI. While PA with 0 in three most significant bytes means direct path, and global broadcast operation with such PA will propagate through ALL SI subtrees, the broadcast with 1 in bit 31 of PA (while bits 28-24 are 0) will cause global broadcast only in subtree, connected to the particular SI in master crate, defined by bits 30,29 (see previous paragraph).*

4.2 Interaction with main DAQ process.

(will be defied soon)

4.3 Subroutines to perform particular actions.

Functions to access modules in fastbus crates are listed in table 3, for modules in CAMAC crates - in table 4.All functions have integer type, and return 0 when completion successful, otherwise returned error code depends on the particular error type.

4.4 Error codes

To get error code explanation one can use call of out_bsierr. To program different

Table 3. Functions for FASTBUS module operations.

ACTION			FUNCTION names and ARGUMENTS
Hardware and software initialization. This function should be called once at the beginning of the program.			F: intini(level) C: intini_(&level)
SI path definition. This function should be called after INTINI. It defines address of SSP for SI path in main crate.			F: ssp_for_si(pa) C: ssp_for_si_(&pa)
Last error explanation. This function prints on stdout and stderr an explanation of recent error code returned			F: out_bsierr() C: out_bsierr_(void)
Bus RESET on the segment, in which module with address PA is member.			F: sspwrb(pa) C: sspwrb_(&pa)
Bus RESET GLOBAL. Issues bus reset on the segment, where module with PA is member and on all segments below .			F: sspwrbg(pa) C: sspwrbg_(&pa)
READ	CSR space	Single word	F: ssprc(pa,sa,buffer) C: ssprc_(&pa,&sa,&buffer)
		Block	F: ssprcb(pa,sa,buffer,nwords) C: ssprcb_(&pa,&sa,&buffer,&nwords)
		Block (end SS=2)	F: ssprc2b(pa,sa,buffer,nwords,nwread) C: ssprc2b_(&pa,&sa,&buffer,&nwords,&nwread)
		Broadcast	F: ssprcm(br_funct,buffer) C: ssprcm_(&br_funct,&buffer)
	DATA space	Single word	F: ssprd(pa,sa,buffer) C: ssprd_(&pa,&sa,&buffer)
		Block	F: ssprdb(pa,sa,buffer,nwords) C: ssprdb_(&pa,&sa,&buffer,&nwords)
		Block (end SS=2)	F: ssprd2b(pa,sa,buffer,nwords,nwread) C: ssprd2b_(&pa,&sa,&buffer,&nwords)
		Broadcast	F: ssprdm(br_funct,buffer) C: ssprdm_(&br_funct,&buffer)
WRITE	CSR space	Single word	F: sspwc(pa,sa,buffer) C: sspwc_(&pa,&sa,&buffer)
		Block	F: sspwcb(pa,sa,buffer,nwords) C: sspwcb_(&pa,&sa,&buffer,&nwords)
		Broadcast	F: sspwcm(br_funct,buffer) C: sspwcm_(&br_funct,&buffer)
	DATA space	Single word	F: sspwd(pa,sa,buffer) C: sspwd_(&pa,&sa,&buffer)
		Block	F: sspwdb(pa,sa,buffer,nwords) C: sspwdb_(&pa,&sa,&buffer,&nwords)
		Broadcast	F: sspwdm(br_funct,buffer) C: sspwdm_(&br_funct,&buffer)

level - defines different levels of initialization. User's programs should use level=0.
pa=PA - primary address, defined as described in the table 3.
sa- secondary address
buffer - is a buffer (integer array) to read data to, or write data from
nwords- number of words to be read or written
nwread - return value. Number of words actualy were read before SS=2 ended operation
br_function - 3 high order bytes are the defined according to the same rules, as for PA, but the lowest byte instead of slot number contains broadcast function code.

behaviour depending on error code, use the next list of returned values:

0	Normal completion.
001-007	SS response of the module being accessed.

Table 4. Functions for CAMAC module operations

ACTION		Function names and arguments.
SSP software initialisation. The call must be made before any other calls to library subroutines. SSP for SI path defiition. Should be performed if FBD is reachable through SI path.		The same intini and ssp_for_si subroutines, as for FASTBUS modules (see table 4)
FBD initialization. This call must preceed all other calls to Struck 320 (FBD).		F: fbd_i(fbd_pa) C: fbd_i_(&fbd_pa)
CAMAC BZ operation.		F: fbd_bz(fbd_pa) C: fbd_bz_(&fbd_pa)
Enable Q-ignore. Responce Q=0 in subsequent operation will not generate error.		F: fbd_qi C: fbd_qi_(void)
Disable Q-ignore. Responce Q=0 in subsequent operations will generate error. This is default at initialization.		F: fbd_q C: fbd_q_(void)
Enable X-ignore. Responce X=0 in subsequent operations will not generate error.		F: fbd_xi C: fbd_xi_(void)
Disable X-ignore. Responce X=0 in subsequent operations will generate error. This is default at initialization.		F: fbd_x C: fbd_x_(void)
Non data-transfer CAMAC operation (CAMAC F-code in the range 8-15,24-31)		F: fbd_n(fbd_pa,f,c,n,a,q,x) C: fbd_n_(&fbd_pa,&f,&c,&n,&a,&q,&x)
READ operations (F=0 - 7)	Single word	F: fbd_r(fbd_pa,f,c,n,a,q,x,buf) C: fbd_r_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buf)
	Autoincrement mode	F: fbd_ra(fbd_pa,f,c,n,a,q,x,buf,nw) C: fbd_ra_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buf,&nw)
	Block transfer mode	F: fbd_rb(fbd_pa,f,c,n,a,q,x,buf,nw) C: fbd_rb_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buf,&nw)
WRITE operations (F=16 - 23)	Single word	F: fbd_w(fbd_pa,f,c,n,a,q,x,buf) C: fbd_w_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buf)
	Autoincrement mode	F: fbd_wa(fbd_pa,f,c,n,a,q,x,buf,nw) C: fbd_wa_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buf,&nw)
	Block transfer mode	F: fbd_wb(fbd_pa,f,c,n,a,q,x,buf,nw) C: fbd_wb_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buf,&nw)
fbd_pa - primary address of FBD module. Rules for it calculation see in the table 3 f - CAMAC function, c - CAMAC crate number, n- slot number, a - subaddress q- q-responce (if not ignored), x - x-responce (if not ignored) buf - buffer to read data in or to write data from nw - number of words to be transferred in autoincrement and block modes		

008(hex) Divide by zero in one of the SSPs.
009(hex) Arithmetic overflow in one of the SSPs.
00A(hex) Unsupported opcode in one of the SSPs.
00B(hex) Block transfer counter limit was exceeded.
00C(hex) Timeout during arbitration cycle on one of the segments.
00D(hex) Timeout during primary address cycle.
00E(hex) Timeout during data cycle.
00F(hex) SSP memory size was exceeded during block transfer.
010(hex) Data cycle without previous address cycle.
011(hex) Unknown Fastbus error.
020(hex) Data buffer in the SSP along the path is to small for transfer.
021(hex) Timeout waiting for response of SSP or SI along the path.
022(hex) Invalid primary address specified.

023(hex) Intermediate module along the SSP path is not SSP.
024(hex) Invalid command for BSI software.
025(hex) SSP or SI along the path is busy.
026(hex) SSP along the path doesn't contain resident BSI software.
027(hex) NTA register read error during multipart transfer of large blocks.
028(hex) Unsupported interface operation.
101(hex) Failure of VBBC-BBFC interface.
102(hex) Power might be off.
2xx(hex) SSP halt code is not BSI_DONE, xx is halt code.

All arguments for function calls are of integer type.

4.5 Where to find libraries.

To use described functions one should link his program with the next libraries:

libsspbsi.a
liblocssp.a
libintsgi.a
libstrlib.a
libutillib.a
libbblib.a

All this libraries are located on the *lemond.phy.bnl.gov* in the directory */usr/people/e852/source/daq/DAQ/lib.* Source codes for these libraries can be found in directory */usr/people/e852/source/daq/DAQ/lib/source,* include files - in */usr/people/e852/source/daq/DAQ/include.* Source directory contains also file *sspbsi_dummy.f* with dummy definition for all functions. This file can be used for debugging programs without connection to real E852 apparatus.

5.On-line data access.

As it was mentioned in section 2, there is the possibility for any process, running on the host SGI computer to access row data, buffered in the shared memory. Such access may be used for low-level data display (for example, MPS group intends to use such access to display chambers hits histograms). The most general way for high level data analysis, with the possibility to many analyzer, running simultaneously on different machines, to join their power, is provided by the DISPATCHER program. DISPATCHER runs on the host SGI420 computer, in the case of on-line data processing,

but may also run on any other computer while of-line processing.

5.1 Dispatcher operation.

Dispatcher has two types of clients - raw data ANALYZERS and DISPLAY (or COOKED data consumer) clients. There is also process, which feeds dispatcher with raw data events - DISPATCHER FEEDER. The last may be used to feed dispatcher with events, read from the tape as well as with events, arriving on-line from E852 apparatus.

As soon, as dispatcher receives event from the feeder, it seeks the available analyzing process. If there is analyzer, waiting for the data, dispatcher sends event to that analyzer, and deletes it from the buffer.If there are not enough analyzers to provide event processing rate adequate to event arriving rate, some events will be dropped. When processing events, arriving from the tape, the incoming event rate is controllable, and in that case dispatcher does not drop events, but stops event feeder, until all events in buffer are processed.

Processed (COOKED) events also come through dispatcher. Every analyzer after completion of event analysis returns to the dispatcher result of the analysis in the form of COOKED event. Upon receiving such event, dispatcher immediately send it to all DISPLAY clients, who are waiting for it. It is also possible for DISPLAY clients to specify particular type of events for given client requests.

So, to say briefly, DISPATCHER ensures every raw event is processed by only ONE analyzer, and EVERY processed event is available to EVERY display client (of course, it does not mean, that every display client actually receives every event - if event arrival rate exceeds ability of display client to consume events, client will receive only such amount of events, which it can consume. Dispatcher does not keep COOKED event any time - it distribute it immediately upon arrival to all waiting clients and then skip it.)

5.2 Library of routines to access data through dispatcher.

The list of functions available at present time for communication with dispatcher is shown in the table 5.The high level code for this functions was written by C.Olchansky, while the bulk of the library was designed by D.P.Weygand. To use this library, one needs to include in the header of his program directive **#include <disIO.h>.** Header disIO.h contains directives to include also ntypes.h and communication.h. All these include files are in the directory */usr/people/e852/source/daq/DAQ/include* on node *lemond.phy.bnl.gov*. The program then should be linked with **disIO.o** file, which is in the directory */usr/people/e852/source/daq/DAQ/lib* on the same machine.

Some of these functions have Error type. That means, that they return value 0 when the call was completed successfully and non-zero if call failed. Function of int type usually return actual number of byte transferred. For the read function this value is less than buffer size, for write function it should be the same, otherwise call should be considered as erratic. In the case of the errors, disIO_... functions print message on the

Table 6. Functions to communicate with DISPATCHER

ACTION		Function type	FUNCTION name and ARGUMENTS
Establish connection with DISPATCHER		Error	disIO_connect(const char*host)
Disconnect from the DISPATCHER		Error	DisIO_disconnect(void)
Send COMMAND to the DISPATCHER		Error	disIO_command(const char*str)
RAW data transfer	READ	int	disIO_readRAW(void*buffer,int bufsize)
	WRITE	int	disIO_writeRAW(const void*buffer,int bufsize)
COOKED data transfer	READ	int	disIO_readCOOKED(void*buffer, int bufsize)
	WRITE	int	disIO_writeCOOKED(const void*buffer,int bufsize)

host - the character string, specifying network address of host, where DISPATCHER running
str - string of character, specifying COMMAND sent to DISPATCHER
buffer - the place to put recieved data or get data to send
bufsize - the muximum size of record to be received and exact data of record to be sent

screen and return negative value.

A list of implemented command for function disIO_command(...) will be provided later.

6.Readout data formats and timing.

As it is seen from the table 1., there are only few type of FASTBUS modules used for data acquisition in E852. They are:

- LeCroy 1881 ADC
- LeCroy 1876 TDC
- Custom made QDC (manufactured by Indiana University)
- Custom made TDC (manufactured by MPS group, BNL)
- Struck STR200 scalers
- ? LATCH modules

There are also few CAMAC modules, used mainly for hardware control (high voltage setting for LGD, transporter movements control,trigger scintillators delays,fast trigger logic), however some information will be read out from CAMAC modules also (transporter position, magnets currents values). The intention is not to read any CAMAC during event readout, as CAMAC is 10 times slower than FASTBUS, but read it only once per spill.

CAMAC modules used in the E852 are:

- Custom made High Voltage Controller,
- LeCroy 4518 delay modules
- LeCroy 4508 Memory Lookup Unit
- ? Transporter Controller,
_ ? Digital Voltmeter module.

To obtain highest possible data rate, the main concern should be about conversion time and readout time. Readout time will be discussed later, as it depends on the data acquisition organisation. CAMAC modules conversion time is not relevant, as we are not going to read CAMAC every event. So we have to pay attention to next conversion times:

- LeCroy 1881 ADC - 15 μsec,
- LeCroy 1876 TDC - about 3 μsec,
- Custom made QDC (IU) - 4 μsec,
- Custom made TDC (MPS) - 600? μsec.

All used FASTBUS modules have sparse data readout and block transfer mode. That means, that event readout time will depend of the amount of information one event contains, and data transfer speed will be limited only by the speed of the FASTBUS MASTER device.

Resulting DAQ limit on data rate will be discussed later in section 10.

Data formats, supplied by employed hardware are shown at the fig.2. As at the present time we have not got LeCroy 1881 and LeCroy 1876 modules, we are going to use LeCroy 1879 and LeCroy 1885 for TCYL test. So data formats for these modules are represented also.

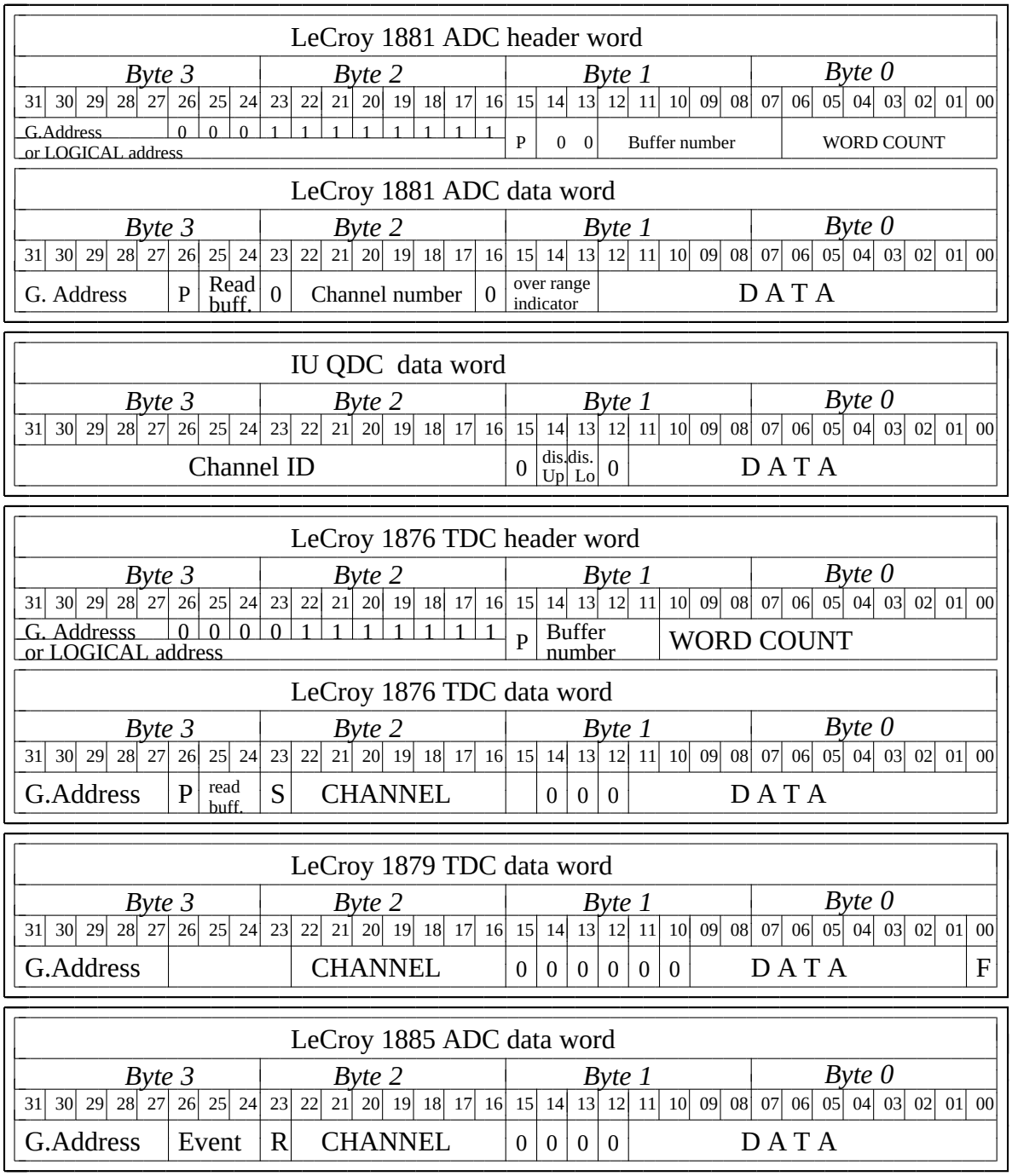
Some comments should be made here:

LeCroy 1881 ADC:

This module has 64 channels, and ability to store up to 8 events in the internal buffer memory. We are not going to explore the multiple event buffering feature of the module, as it could give us saving only on conversion time (next event digitizing could be launched before completion of event readout). In our case the full event readout time is about 500 msec, so saving of 15 msec on conversion hardly worth of serious complication of software and triggering system to explore internal event buffering advantage.

If appropriate mode of operation is chosen, 1881 ADC records in its internal memory only output from those channels, where signal exceed set threshold. So event can contain less than 64 words of data. In the internal memory each event buffer has header word,which format is shown on the picture. WORD COUNT - the amount of

Fig 2. Readout data formats



words in the following event buffer (including header word itself). Buffer number shows the position of event within module memory. As we are not going to use the buffering events in the module,this value will be 0 all the time. P means parity, and this bit will provide even number of "1" in the whole word. G.Address - GEOGRAPHICAL ADDRESS of module. In the case of using logical addressing, bits 31-16 will be occupied by LOGICAL ADDRESS.

Data word (see the picture) contains 13 bit data of digitized signal (though the ADC resolution is claimed 12 bit, this appeared 12 bit above pedestal, so together with pedestal value may exceed 4096 counts),and 3 bits over range indicator - all these 3 bits are set simultaneously in the case signal exceed ADC range. Read buffer bits are the least bits of buffer number. For us they will be 0 all the time. P bit makes number of "1" in the data word even.

IU QDC:

Module contains 32 QDC channels.The conversion time of this QDC is about 5 μsec, so there was no reason to worry about internal multiple event buffer. There is no data header also, the amount of words in the event is determined by the FASTBUS MASTER with the receiving END OF DATA signal (SS=2). Data word along with 12 bits of digitized signal, contains 2 bits, showing if the signal exceeds preset LOW LEVEL and HIGH LEVEL discriminators thresholds, and 16 bit channel ID. Channel ID are the entirely programmable value, it does not contain even geographical address. So, if QDC are not initialized (in the sense of recording channels IDs in the special designated registers in the control area), data, read from QDC would not be usable, as it would not contain any channel reference.

The IU QDCs have the feature of MULTIBLOCK data readout. This means, that in the case of MULTIBLOCK mode of operation chosen, all the data from entire FASTBUS crate are transferred by single READ command. Data transfer begins from the leftmost module in the crate (its address should be the primary address of READ command), and automatically proceed to the next module upon exhausting data buffer in the current module. So there will not be any visible boundaries in the data structure between different modules in the same crate.

IU QDC memory contains 3 different event buffers. First buffer contains data for all channels, second buffer - data only for the channels, which signal value exceed LOW LEVEL discriminator threshold, and third buffer - data for the channels, where signal exceed HIGH LEVEL discriminator threshold. Looking on the data, it is impossible to distinguish between different buffers,and there is no need to know it, as the signal values, recorded in any buffer will be the same. The only concern about using appropriate buffer should be taken when writing code for data acquisition, as the pedestal values could be read only from the first buffer, while normal event data acquisition should use second buffer. Third buffer is intended for usage by trigger processor, so DAQ may use it only for tuning and debugging.

LeCroy 1876 TDC:

This module also feature multiple event buffer, which has no sense for us, as the conversion time is only about 3 msec (500 ns + 30 ns/hit). This is multihit, 96 input TDC. Data format for this TDC is similar to that of 1881 ADC - there is also header word. The buffer number will be all the time 0 in our case. Bit 23 in data word (S) shows the sign of

the edge: 0 for leader and 1 for trailer. As the 1876 is multihit TDC, each event can contain more than one (up to 16) words for the same channel.

LeCroy 1879 TDC:

The module is multihit TDC. It does not contain multievent buffer (which is not important for us, as we are not using it). There is no header word also. There can be few words for each channel, as TDC is able to record few hits. Even in the case of one hit,two words for corresponding channel will be recorded - one containing time of leading edge of the signal, another - trailing edge. However, information for leading edges is read from DSR with secondary address 0, while for trailing edge - from DSR register with address 1. Whether to read or not information about trailing edges is defined by DAQ configuration (there is corresponding resource for this). However, in recorded data there will be no information about type of the edge. To resolve this ambiguity, the convention about format of data block for 1879 module was adopted, that in the case of recording information about both edges, data block will consist of two parts, separated with word FFFF(hex), which may never occur in data itself.

TDC has 9 bit resolution, however there is 10th bit in DATA, representing time measurement. This bit (F) on the picture codes the faze of clock generator in the time of arrival stop signal. It is not equivalent to the two times higher resolution, as we know the stop signal arrival time with the accuracy of 1/2 of clock period, but, the arrival time of the input signal is known with ambiguity of the whole clock period.

LeCroy 1885 ADC:

Module has ability to store up to 8 events. So there is EVENT field in data word, which shows the position of event in this buffer. For us it will be 0 all the time. Module has also expended dynamic rage of the input signals. It is made by changing conversion scale (bin width). If input signal exceed 175 pC, the bin width automatically changes from 50 fC per count to 400 fC per count. It is also possible to choose constant bin width with CSR0 settings. Data word contains bit R (bit 23), which shows the scale (range) of digitized signal. It is very inconvenient for zero suppression (module has no sparse data readout mode, it output all 96 channels even only pedestals are recorded in them). However, it is very time-consuming operation to change the recorded data format (while SSP allows to transfer data from FASTBUS modules into SSP memory with the speed of 5 Megawords/sec, any SSP operation with single word takes about 1.6 μsec. So it is not decided now, if this format will be changed in the data record.

MPS TDC:

.....

7.DAQ data formats

7.1 DAQ raw data format (before event builder)

There are 3 different data formats, used for data storage in E852. First one is the raw data format, as they are recorded by the main DAQ process, which transfers data from the SSPs into SGI shared memory. As the speed of data transfer is critical, this process copies content of SSPs memory as it is, without any transformation. Within this format there will be 3 different types of records: BEGIN OF RUN record, SPILL record and END OF RUN record. Formats for each of these records are shown on the pictures (fig.3,4).

Some comments to the pictures:

1.Current version of DAQ software anticipates maximum number of SSPs in the system not exceeding 11 (one primary accessed through BBFC, and up to 10 secondary ones). If real number of the SSPs in the system will be smaller, values SSP_offset[n] for $n \geq$ real number of SSPs will be set to 0. Also in the case of fatal error in the SSP redout corresponding SSP_offset will be set to -1.

2.What is called MODULE block in reality may contain data from more than one module.For example, data from LGD will be read in MULTIBLOCK TRANSFER mode, which provides redout of all data from one crate within single operation. In such case the value "module geographical address" (offset 1 in MODULE block, see fig.5) will be equal to the geographical address of first module, which started redout).

3.According to FASTBUS definitions 16 MSB of CSR0 of each FASTBUS module represent module ID, which is unique for every type of FASTBUS device. So SSPs BRUN blocks will contain information about what type of module is placed in each slot.

7.2 Tape raw data format.

7.3 Dispatcher data formats.

10.Data rates.

Let us estimate average size of the event. LGD contributes the most significant part of it. The amount of the LGD data may be estimated in two ways. First way based on physics, leads to the next approaches:

- Average number of clusters (photons) in one event depends on the trigger logic, and is of the order of 3.5 - 5 clusters/event.

Fig 3. DAQ RAW DATA FORMAT (format 1).

BEGIN of RUN in DAQ RAW DATA FORMAT

Offset	Word meaning	Value
0	Record length (number of long words)	length
1	Record type	0 for BRUN
2	Run number	runNo
3	Start of run time	StartOfRunTime
4	Offset of SSP0 BRUN block	SSP_offset[0]=15
5	Offset of SSP1 BRUN block	SSP_offset[1]
.....	.....	.....
14	Offset of SSP10 BRUN block	SSP_offset[10]
15	SSP0 BRUN block. Contain hardware configuration for correspondent SSP at the beginning of run. Format of this block see at the separate picture.	
.....	.....	
SSP_offset[10]	SSP10 BRUN block. Contain hardware configuration for correspondent SSP at the beginning of run. Format of this block see at the separate picture.	

SPILL RECORD in DAQ RAW DATA FORMAT

Offset	Word meaning	Value
0	Record length (number of long words)	length
1	SPILL number	spillNo
2	Run number	runNo
3	Time stamp	time1
4	Number of events in the spill (excluding BSPI & ESPI)	nevents
5	Offset of SSP0 SPILL block	SSP_offset[0]=16
6	Offset of SSP1 SPILL block	SSP_offset[1]
.....	.....	.....
15	Offset of SSP10 SPILL block	SSP_offset[10]
16	SSP0 SPILL block. Format of this block is shown at the separate picture.	
.....	.....	
SSP_offset[10]	SSP10 SPILL block. Format of this block is shown at the separate picture.	

END of RUN in DAQ RAW DATA FORMAT

Offset	Word meaning	Value
0	Record length (number of long words)	length
1	Record type	-1 for ERUN
2	Run number	runNo
3	End of run time	EndOfRunTime
4	Total size of run (kilobytes, including BRUN & ERUN)	runSize
5	Number of spills in this run	nspills
6	Number of events in the run (excluding BSPI & ESPI)	nevents
7	Offset of SSP0 ERUN block	SSP_offset[0]=18
8	Offset of SSP1 ERUN block	SSP_offset[1]
.....	.....	.....
17	Offset of SSP10 ERUN block	SSP_offset[10]
18	SSP0 ERUN block. Contain hardware configuration for correspondent SSP at the end of run. Format of this block see at the separate picture.	
.....	.....	
SSP_offset[10]	SSP10 ERUN block. Contain hardware configuration for correspondent SSP at the end of run. Format of this block see at the separate picture.	

- In the average one cluster occupies about 10 blocks of lead glass. It depends on the cluster energy and threshold for the signal recording, however transversal attenuation of the light amount with the distance from cluster center occur very fast, so one

Fig 4. DAQ RAW DATA FORMAT (format 1), continuation.

SSP BRUN/ERUN blocks in DAQ RAW DATA FORMAT

Offset	Word meaning	Value
0	Length of block(number of long words)	length
1	SSP ID (full geographical addresss)	SSPadr
2	Number of modules found	
3	Geographical address of 1st module	
4	CSR0 of 1st module	
5	Geographical address of 2nd module	
6	CSR0 of 2nd module	
.....	.....	

SSP spill block in DAQ RAW DATA FORMAT

Offset	Word meaning	Value
0	Block length (number of long words)	length
1	SSP ID (full geographical addresss)	SSPadr
2	Evets in spill (excluding BSPI & ESPI)	nevents
3	Number of events with errors.	nbad
4	Spill error word.Bits, corresponding slots with at least one error reading during the spill are set.	badSlots
5	BSPI data bank	
.....	EVENT data bank	
.....	.....	
.....	EVENT data bank	
.....	ESPI data bank	

BSPI,EVENT & ESPI data bank in DAQ RAW DATA FORMAT

Offset	Word meaning	Value
0	Total length of data bank (number of long words).	length
1	EVENT number (0 for BSPI, -1 for ESPI)	eventNo
2	Trigger type	trigger
3	Event error word (set bits correspond error redout slots)	badSlots
4	MODULE block	
	MODULE block	
.....	.....	
.....	.....	
	MODULE block	
	MODULE block	

MODULE block in DAQ RAW DATA FORMAT

Offset	Word meaning	Value
0	Length of module block(number of long words)	length
1	Module geographical address	
2	SSP error word if FASTBUS error occured during module redout	
3	Module redout data. Formats of these data depend on the particular module type, and are discussed in the section READOUT DATA FORMATS	

photon almost never produce cluster, occupying more than 16 blocks, independently of energy and registration threshold.

So we can estimate average amount of LGD data as 5(mu10(mu4(eq200 bytes per event.

Another approach is based on the affordable noise level. Definitely, due to the any kind of noises, the pedestal value (the QDC output in the absence of any signal) will fluctuate. If we want to register the smallest possible signals, we will try to set registration threshold as low, as possible. If we set it close enough to the pedestal,we will record a lot of signals just due to noises (if we set threshold in each channel equal to pedestal, we can expect 50% of all channels in each event will have signal,exceeding threshold. So the question is, what amount of noises could we afford. The reasonable approach seems to be 10% of all channels, which means about 300 words or 1200 bytes of data per event.

Drift chambers??????????????????????????

The expected number of useful events per spill may be estimated on the base of beam rate 2*10, and all-neutral cross-section ,which gives us the rate 720 all neutral events/spill. The number of events with charge particles will be much greater (about 15000/spill with multiplicity < 3), however including into the trigger criteria demand of large number of photons in the same event can reduce this value to few thousands/spill (as the top case of low charge particle and high photon multiplicity is all neutral event). So we can estimate, that reasonable goal for data acquisition system is the ability to accept about 1000 events/spill. This gives us the data rate about 2-3 Mbytes/spill or about 1 Mbytes/sec.