E852 Software note

November 8, 1992

LIBRARY OF ROUTINES TO ACCESS FASTBUS AND CAMAC MODULES DIRECTLY FROM E852 DAQ COMPUTER

A.Ostrovidov

The purpose of this note is to give a brief description of the library that can provide direct access to E852 hardware from DAQ computer - i.e., from Silicon Graphics with installed VMEbus-Branch Bus Controller (VBBC) inside the SGI VME slot and with Branch Bus-Fastbus Controller (BBFC) inside the Fastbus master crate. This library may be used by anyone who is going to write the programs accessing any modules in E852 Fastbus or CAMAC crates independently of the main *DAQ* processes.

This library was written by R.Hackenburg at BNL. Small changes have been made in the code which provides access to modules through "SSP path" (see below). I have added also support for ordinary Segment Interconnects ("SI path"). Access code for CAMAC crates ("FBD path") is the same as in the original R.Hackenburg’s library without any changes. SGI version of this library uses, in its turn, the BBLIB library of R.Imossi for VBBC interface specific actions.

Which Modules Can Be Accessed With This Library?

The hardware configuration of E852 experiment will include (Fig.1):

- SGI Power Series computer running UNIX (i.e., IRIX 4.0.5) with VBBC board connected to VMEbus. Thus, all programs which will access E852 hardware should be UNIX-compatible.

- Master Fastbus crate with BBFC board. This board communicates with VBBC via Branch Bus. Master SSP and at least one SI reside in this crate. Path from host

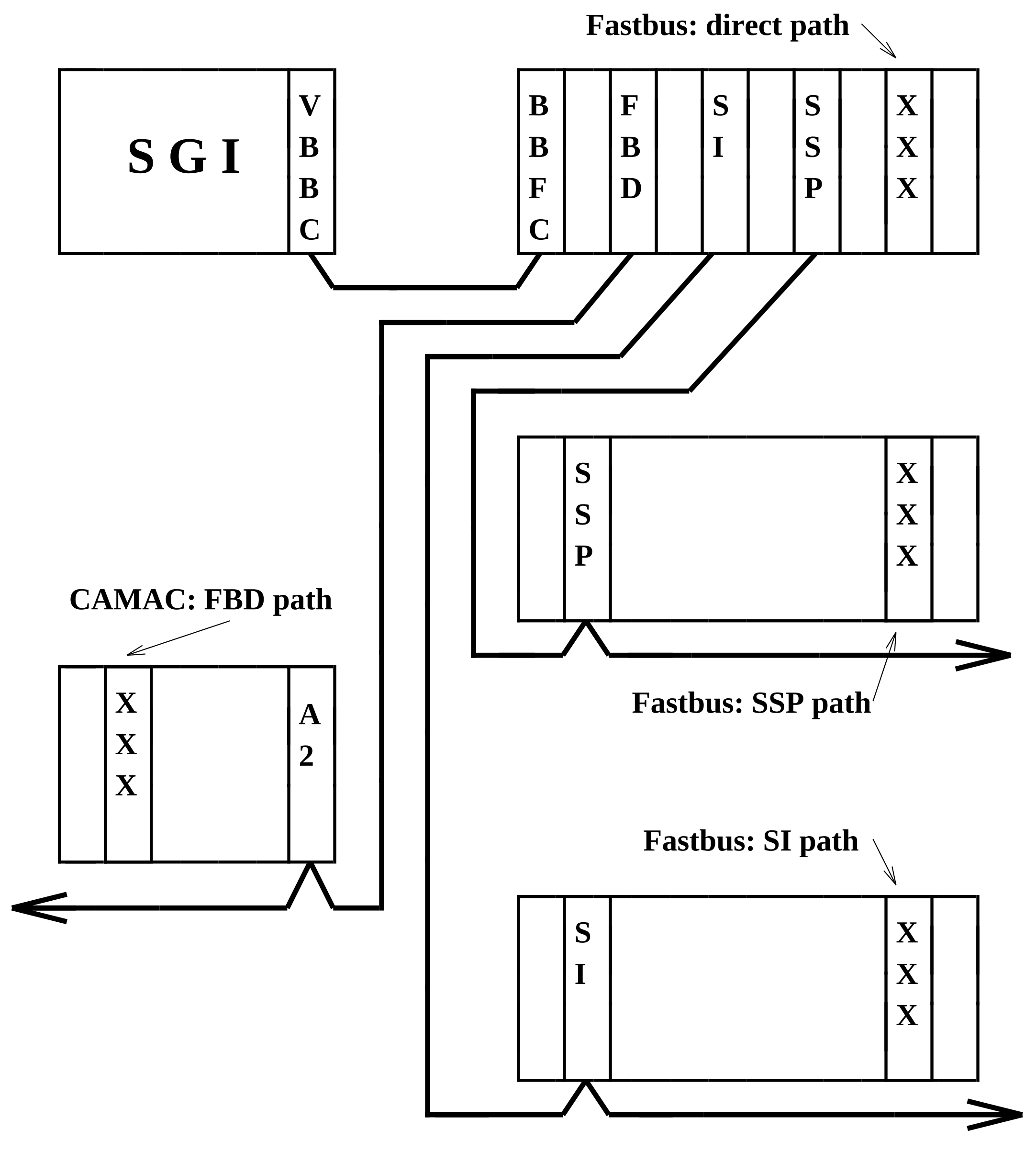

Fig. 1

computer to any module in this crate will be referenced below as "direct path".

- Fastbus crates with SSPs. These slave SSPs are connected to master SSP with Fastbus Cable Segment. Any module in the crate with slave SSP can be reached from host over "SSP path".

- Fastbus crates with SIs. These SIs are connected to SI in the master crate with another Fastbus Cable Segment. Any module in the crate with SI can be reached from host over "SI path".
- CAMAC crates with A2 type controller. These crates are managed by Fastbus Branch Driver (FBD) - Struck model 320. FBD can be placed either into master crate or into slave crate with SSP or into slave crate with SI but position in the master crate is preferable. Path from host to any module in the CAMAC crate will be referenced as "FBD path".

Any other configuration of crates is not supported currently. For example, it's impossible to reach Fastbus module in the "third level" crate (i.e., in the crate connected through SSP or SI path not to a master crate but rather to the "second level" crate). Anyway, we are not going to use such exotic configurations in the E852 experiment.

Primary Address Convention

Practically all library routines require parameter PA - primary address of the module as seen from the host computer. Direct, SSP and SI paths are based on the same set of routines, and the choice of the path to the module is defined by the 32-bit integer PA value. The following rules are used in definition of PA.

1. Module is on the direct path, i.e. inside the master crate in the slot N.

PA =	0	0	0	N

Three high order bytes should be zero for this case.

Example: module inside the master crate in the slot 0E(hex). Use PA = 0000000E(hex) = 14(dec).

2. Module is on the SSP path, i.e. in the slot N1 in the crate with SSP whose cable address is N2 (set with DIP switch). Cable is connected to SSP in the slot N3 inside the master crate.

PA =

0	N3	N2	N1

The high order byte should be zero.

Example: module in the slot 0E(hex), SSP in this crate has cable address 0A(hex), master SSP is in the slot 7(hex) in the master crate. Use PA = 00070A0E(hex) = 461326(dec).

3. Module is on the SI path, i.e. in the slot N1 in the crate with SI whose cable address is N2 (set with DIP switch). Cable is connected to SI in the master crate.

PA =

N2	0	0	N1

The high order byte is non-zero (two middle bytes are ignored in this case).

Example: module in the slot 0E(hex), SI in this crate has cable address 0A(hex). Use PA = 0A00000E(hex) = 167772174(dec).

4. Module is on the FBD path, i.e. in the slot N in the CAMAC crate C. These values are used directly as parameters in the CAMAC specific functions calls. In addition, parameter FBD_PA defines the primary address of FBD module in accordance with the rules described above.

Notes:

⇒ SSPs in the master crate shouldn't be placed in slot 0.

⇒ SSPs and SIs must have cable addresses in the range 1-31.

⇒ PA with two zero high order bytes (i.e., rule 2 with N3=0) will refer to slave SSP memory rather than to any module in Fastbus crate. Be careful to avoid such PA values - the contents of SSP memory can be destroyed.

⇒ In the unlikely case of a few SIs in the master crate (each with separate cable segment subtree) these SIs will be consequently enumerated: SI with the lowest address in the master crate will be #0, SI with the next address will be #1 and so on. This number will occupy 3 high order bits in the high order byte of PA while N2 (see rule 3) will occupy 5 lowest bits of this byte.

⇒ Before you start your program all SSPs should be loaded with necessary code, routing tables in all SIs have to be initialized and interface should be reset. Main *DAQ* process takes care about this stuff. If this is not the case (usually, after switching power off and on) then it's necessary to start *DAQ* first and choose "INIT" option from *DAQ* menu or, alternately, run *DAQinit* program.

Fastbus And CAMAC Common Functions

1.

Fortran:

```
INTEGER INTINI(LEVEL)
INTEGER LEVEL
```

C:

```
int intini_(&level)
int level;
```

This function should be called once at the beginning of the program before any other calls. It performs necessary software and hardware initialization. LEVEL defines different levels of initialization. User's programs should run with LEVEL=0.

2.

Fortran:

```
INTEGER FUNCTION SSP_FOR_SI(PA)
INTEGER PA
```

C:

```
int ssp_for_si_(&pa)
int pa;
```

This function defines which SSP in the master crate to use as mediator between host computer and modules on SI path. Host can not communicate with SIs directly due to limitations in the BBFC (this board provides only 5 bits of geographical address instead of necessary 32 bits for SIs). Every time the program wants to access module on the SI path host downloads necessary information into chosen SSP and ask it "a favor" to do this job (this SSP is already in the master crate and, thus, it avoids the bottleneck of BBFC). PA is SSP address in the master crate. This function should be called once at the beginning of the program immediately after call to INTINI. In case of CAMAC, program should use this function if FBD is on SI path.

3.

Fortran:

```
INTEGER FUNCTION OUT_BSIERR()
```

C:

```
int out_bsierr_(void)
```

Print on stdout and stderr an explanation of the most recent error in case of non-zero return from any function in this library.

Functions For Fastbus Modules

PA	is module’s primary address which is subject to described rules.
SA	is a secondary address for Fastbus operation.
BUFFER	is a buffer to read data to or write data from.
NWORDS	is a number of words to read/write.

4.

Fortran:

```
INTEGER FUNCTION SSPRC(PA,SA,BUFFER)
INTEGER PA,SA,BUFFER
```

C:

```
int ssprc_(&pa,&sa,&buffer)
int pa,sa,buffer;
```

Read a single word from CSR space into BUFFER.

5.

Fortran:

```
INTEGER FUNCTION SSPRCB(PA,SA,BUFFER,NWORDS)
INTEGER PA,SA,BUFFER(*),NWORDS
```

C:

```
int ssprcb_(&pa,&sa,&buffer,&nwords)
int pa,sa,buffer[ ],nwords;
```

Read a block of NWORDS words from CSR space into BUFFER.

6.

Fortran:

```
INTEGER FUNCTION SSPRCM(BR_FUNCT,BUFFER)
INTEGER BR_FUNCT,BUFFER
```

C:

```
int ssprcm_(&br_funct,&buffer)
int br_funct,buffer;
```

Broadcast read of a single word from CSR space into BUFFER. BR_FUNCT is subject to the same rules as PA, only with slot number (the lowest byte) replaced by broadcast function code. Only local broadcasts in the crate specified by 3 high order bytes are supported.

7.

Fortran:

```
INTEGER FUNCTION SSPRD(PA,SA,BUFFER)
INTEGER PA,SA,BUFFER
```

C:

```
int ssprd_(&pa,&sa,&buffer)
int pa,sa,buffer;
```

Read a single word from data space into BUFFER.

8.

Fortran:

```
INTEGER FUNCTION SSPRDB(PA,SA,BUFFER,NWORDS)
INTEGER PA,SA,BUFFER(*),NWORDS
```

C:

```
int ssprdb_(&pa,&sa,&buffer,&nwords)
int pa,sa,buffer[ ],nwords;
```

Read a block of NWORDS words from data space into BUFFER.

9.

Fortran:

```
INTEGER FUNCTION SSPRDM(BR_FUNCT,BUFFER)
INTEGER BR_FUNCT,BUFFER
```

C:

```
int ssprdm_(&br_funct,&buffer)
int br_funct,buffer;
```

Broadcast read of a single word from data space into BUFFER. BR_FUNCT is subject to the same rules as PA, only with slot number (the lowest byte) replaced by broadcast function code. Only local broadcasts in the crate specified by 3 high order bytes are supported.

10.

Fortran:

```
INTEGER FUNCTION SSPWC(PA,SA,BUFFER)
INTEGER PA,SA,BUFFER
```

C:

```
int sspwc_(&pa,&sa,&buffer)
int pa,sa,buffer;
```

Write a single word to CSR space from BUFFER.

11.

Fortran:

```
INTEGER FUNCTION SSPWCB(PA,SA,BUFFER,NWORDS)
INTEGER PA,SA,BUFFER(*),NWORDS
```

C:

```
int sspwcb_(&pa,&sa,&buffer,&nwords)
int pa,sa,buffer[ ],nwords;
```

Write a block of NWORDS words to CSR space from BUFFER.

12.

Fortran:

```
INTEGER FUNCTION SSPWCM(BR_FUNCT,BUFFER)
INTEGER BR_FUNCT,BUFFER
```

C:

```
int sspwcm_(&br_funct,&buffer)
int br_funct,buffer;
```

Broadcast write of a single word to CSR space from BUFFER. BR_FUNCT is subject to the same rules as PA, only with slot number (the lowest byte) replaced by broadcast function code. Only local broadcasts in the crate specified by 3 high order bytes are supported.

13.

Fortran:

```
INTEGER FUNCTION SSPWD(PA,SA,BUFFER)
INTEGER PA,SA,BUFFER
```

C:

```
int sspwd_(&pa,&sa,&buffer)
int pa,sa,buffer;
```

Write a single word to data space from BUFFER.

14.

Fortran:

```
INTEGER FUNCTION SSPWDB(PA,SA,BUFFER,NWORDS)
INTEGER PA,SA,BUFFER(*),NWORDS
```

C:

```
int sspwdb_(&pa,&sa,&buffer,&nwords)
int pa,sa,buffer[ ],nwords;
```

Write a block of NWORDS words to data space from BUFFER.

15.

Fortran:

```
INTEGER FUNCTION SSPWDM(BR_FUNCT,BUFFER)
INTEGER BR_FUNCT,BUFFER
```

C:

```
int sspwdm_(&br_funct,&buffer)
int br_funct,buffer;
```

Broadcast write of a single word to data space from BUFFER. BR_FUNCT is subject to the same rules as PA, only with slot number (the lowest byte) replaced with broadcast function code. Only local broadcasts in the crate specified by 3 high order bytes are supported.

16.

Fortran:

```
INTEGER FUNCTION SSPWRB(PA)
INTEGER PA
```

C:

```
int sspwrb_(&pa)
int pa;
```

Issues Reset Bus on the segment in which the module with address PA is a member.

17.

Fortran:

```
INTEGER FUNCTION SSPWRBG(PA)
INTEGER PA
```

C:

```
int sspwrbg_(&pa)
int pa;
```

Issues Reset Bus Globally, i.e., on the segment in which the module with address PA is a member and on all segments below it in subtree.

Notes:

⇒ Broadcast block transfers are not implemented.

⇒ Block transfers with SS=2 are not implemented and generate error if attempted. Let me know if this creates any problems.

Functions For CAMAC Modules

Descriptions of this functions were taken from R.Hackenburg's note "BNL access routines for the SSP", MPS note 86, December 1, 1989.

FBD_PA	is a primary address of FBD board in accordance with described rules.
C	is a CAMAC crate number (set with DIP switch in A2 type controller).
N	is a crate slot address of accessed module.
A	is a subaddress inside the module.
F	is a CAMAC operation to perform.
BUFFER	is a buffer to read data to or write data from.
NWORDS	is a number of words to read/write.
Q	is a status of Q line following or accompanying the operation.
X	is a status of X line following or accompanying the operation.

Only 3 low order bytes (24 lowest bits) of data words are used in read/write operations.

18.

Fortran:

```
INTEGER FUNCTION FBD_I(FBD_PA)
INTEGER FBD_PA
```

C:

```
int fbd_i_(&fbd_pa)
int fbd_pa;
```

Initializes a Struck 320 (FBD), at Fastbus primary address FBD_PA. This call must precede all other FBD calls.

19.

Fortran:

```
INTEGER FUNCTION FBD_BZ(FBD_PA)
```

```
INTEGER FBD_PA
```

C:

```
int fbd_bz_(&fbd_pa)
int fbd_pa;
```

Issues a CAMAC BZ via FBD at Fastbus primary address FBD_PA.

20.

Fortran:

```
INTEGER FUNCTION FBD_N(FBD_PA,F,C,N,A,Q,X)
INTEGER FBD_PA,F,C,N,A
LOGICAL Q,X
```

C:

```
int fbd_n_(&fbd_pa,&f,&c,&n,&a,&q,&x)
int fbd_pa,f,c,n,a
unsigned char q,x
```

Performs CAMAC non-data-transfer operation, i.e., operation with F-code in the range 8 to 15 and 24 to 31. Returns Q and X if they are not ignored.

21.

Fortran:

```
INTEGER FUNCTION FBD_R(FBD_PA,F,C,N,A,Q,X,BUFFER)
INTEGER FBD_PA,F,C,N,A,BUFFER
LOGICAL Q,X
```

C:

```
int fbd_r_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buffer)
int fbd_pa,f,c,n,a,buffer
unsigned char q,x
```

Performs single CAMAC read operation (F-code 0 to 7) into BUFFER. Also returns Q and X if they are not ignored.

22.

Fortran:

```
INTEGER FUNCTION FBD_RA(FBD_PA,F,C,N,A,Q,X,BUFFER,NWORDS)
INTEGER FBD_PA,F,C,N,A,BUFFER(*),NWORDS
LOGICAL Q,X
```

C:

```
int fbd_ra_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buffer,&nwords)
int fbd_pa,f,c,n,a,buffer[ ],nwords
unsigned char q,x
```

Performs CAMAC auto increment read operation (F-code 0 to 7) of NWORDS into BUFFER. Also returns Q and X if they are not ignored.

23.

Fortran:

```
INTEGER FUNCTION FBD_RB(FBD_PA,F,C,N,A,Q,X,BUFFER,NWORDS)
INTEGER FBD_PA,F,C,N,A,BUFFER(*),NWORDS
LOGICAL Q,X
```

C:

```
int fbd_rb_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buffer,&nwords)
int fbd_pa,f,c,n,a,buffer[ ],nwords
unsigned char q,x
```

Performs CAMAC block read operation (F-code 0 to 7) of NWORDS into BUFFER. Also returns Q and X if they are not ignored.

24.

Fortran:

```
INTEGER FUNCTION FBD_W(FBD_PA,F,C,N,A,Q,X,BUFFER)
```

```
INTEGER FBD_PA,F,C,N,A,BUFFER
LOGICAL Q,X
```

C:

```
int fbd_w_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buffer)
int fbd_pa,f,c,n,a,buffer
unsigned char q,x
```

Performs single CAMAC write operation (F-code 16 to 23) of word BUFFER. Returns Q and X if they are not ignored.

25.

Fortran:

```
INTEGER FUNCTION FBD_WA(FBD_PA,F,C,N,A,Q,X,BUFFER,NWORDS)
INTEGER FBD_PA,F,C,N,A,BUFFER(*),NWORDS
LOGICAL Q,X
```

C:

```
int fbd_wa_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buffer,&nwords)
int fbd_pa,f,c,n,a,buffer[ ],nwords
unsigned char q,x
```

Performs CAMAC auto increment write operation (F-code 16 to 23) of NWORDS from BUFFER. Returns Q and X if they are not ignored.

26.

Fortran:

```
INTEGER FUNCTION FBD_WB(FBD_PA,F,C,N,A,Q,X,BUFFER,NWORDS)
INTEGER FBD_PA,F,C,N,A,BUFFER(*),NWORDS
LOGICAL Q,X
```

C:

```
int fbd_wb_(&fbd_pa,&f,&c,&n,&a,&q,&x,&buffer,&nwords)
int fbd_pa,f,c,n,a,buffer[ ],nwords
```

```
unsigned char q,x
```

Performs CAMAC block write operation (F-code 16 to 23) of NWORDS from BUFFER. Returns Q and X if they are not ignored.

27.

Fortran:

```
SUBROUTINE FBD_Q
```

C:

```
void fbd_q_(void)
```

Disables Q-ignore. This is default at initialization.

28.

Fortran:

```
SUBROUTINE FBD_QI
```

C:

```
void fbd_qi_(void)
```

Enables Q-ignore. Response Q=0 will not generate an error.

29.

Fortran:

```
SUBROUTINE FBD_X
```

C:

```
void fbd_x_(void)
```

Disables X-ignore. This is default at initialization.

30.

Fortran:

```
SUBROUTINE FBD_XI
```

C:

```
void fbd_xi_(void)
```

Enables X-ignore. Response X=0 will not generate an error.

Return Value

All functions of integer type return 32-bit integer BSI_STATUS - the status word of the last operation. It should be zero in case of success. If it is non-zero then subsequent call of out_bsierr() function gives detailed explanation of last error. Possible values of BSI_STATUS are:

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

Error codes 1-7,B,D,20,22,23 may be caused by the wrong parameters values in the function call or illegal operation with accessed module in the user's program. Other error codes are unlikely and may indicate problems with hardware, SSP software bugs or that *DAQinit* was not run after power up.

Final Notes

If your program uses any of these routines it should be linked together with the following libraries:

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

All these libraries can be found on lemond.phy.bnl.gov in the directory /usr/people/e852/source/daq/DAQ/lib. As far as I know this disk is NFS mounted to IU and ND SGI computers. Source codes for these libraries are in /usr/people/e852/source/daq/DAQ/lib/source, necessary include files - in /usr/people/e852/source/daq/DAQ/include. Source directory contains also file sspbsi_dummy.f with dummy definitions for all functions. This file can be useful at the development stage to avoid "Undefined symbols" messages from linker and to test program in the simple "simulation" mode without any hardware. In this mode program can be tested on any computer (including VAX) with understanding that it has to be ported later on E852 DAQ Silicon Graphics.

Code supporting direct and SSP paths was intensively tested. It seems that now there are no bugs in it. Code for FBD path was not tested (we haven't FBD yet). On the other hand, this part of library was tested by R.Hackenburg and no changes have been made after that. Support for SI path was written from scratch and never tested (we haven't also SIs up to now). Probably, this library will require some changes in the future, but those changes will not affect the user's program (except of necessity to relink it with the upgraded library).