(Version 2.0 of 88-62)
(Version 2.0 of 88-12)

ALEPH 88-146
DATACQ 88-21
A. Miotto et al.
14.11.1988

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

Aleph Event Builder FASTBUS library v2.0

A.Castro, A. Miotto.

This paper has: 36 pages ~~figures~~ ~~annexes.~~
If you wish to receive it, please send your request to:
A. Mazzari - EP

Name: ______________________ Div.: ____________
I would like to receive 1 copy of the paper: with annex □ without annex □
Title (or ALEPH No.): ______________________
Author(s): ______________________
Date: ____________ Signature: ______________________

rsion 2.0 of 88-62)
rsion 2.0 of 88-12)

ALEPH 88-146
DATACQ 88-21
A. Miotto et al.
14.11.1988

R LA RECHERCHE NUCLÉAIRE
OR NUCLEAR RESEARCH

Physique des Particules
r Particle Physics

Builder
rary v2.0

Miotto.

low multi-user utilization
es on the Aleph Event
rating system.

A.Castro, A. Miotto.
AL0VOL::MIOTTO at CERN
2.0
3 November 1988
First update

sion:

RELEASE

MODC, FMODD, FBFIM, FRRTB, FWRTB.

FCAS2C and FCAS2D (syntax changed).
FALSE.

ws FASTBUS actions to be

ltaneous users and most of

the coprocessor under the

, and is called by the library

t management and interrupt

void software overhead: the

s.

ndard defined by U.S. NIM

he coprocessor and to keep

and C languges. Calling

value; the FASTBUS error

ers passed by reference.

executed (for example from a

'y

work properly.

_open before any other call

reated and initialised.

S environments.

by the user. In case of values

on the standard error output

iting.

TRAN entry points for SR

ion per receiver block number

mplementation are defined in

n the short form, excepted the

'e arosen with the standard C

fb_close are used instead.

hould be placed in the DEFS

'. If the make utility is loaded,

led with the command:

n

e of several FASTBUS calls

environments if the user does

V2.1 some situations arose in

ve been fixed with OS-9 V2.2.

JS are stopped and then type:

oftware

he only

ımmary

in this

tandard

is added

r. They

STBUS

named

neters to

ase 2.0.

us some

rocessor

permit a

ry is not

is could

pts). For

ASTBUS session will receive a FEFTL error and will

descriptor (revision 4) are installed in your EB, you will
fb.l and the updated include file cfbdef.h. No such

eters;

st block transfer. Please note that the coprocessor does not
occurs in the synchronous part of the instruction, and that
in any way. The following fragment of code shows a

```
k (FBDEID, prim,sec, FBVAR, buffer, count);
, &actual_count);
 == actual_count) actual_count = 0;
```

_find_supplementary and reading the field
yte_counter. Here the correct value is stored on any
ccurred the content is meaningless. In any case
to know the length of data transferred using a firdb

d-modify-write a FASTBUS word;
t message; this routine requires the EB Memory Manger
as to be linked with the emmlib.l library;
write_route_table_block: these routines do not
rs, but emulate them by executing a loop of frrt and

een corrected (the swapped value were not rturned to the

y fcasc2 and fcasd2) the syntax has changed;
ignal library signals.l;
l by fb_fir_connect (except the message address) are
s been called, and by reference if the FORTRAN one has

s now FB_FALSE as defined in the standard. It is anyway
ine to set it explicitely to the desired value;
ter has been introduced. The time-out value cannot be
assume the values FB_WTT_SHORT (~10 ms, default) and

[1] U.S. NIM Committee - FASTBUS standard routines - March 1987 DOE/ER-0325

FASTBUS ROUTINES

1. ENVIRONMENT MANAGEMENT

FB_OPEN Open a FASTBUS session.

C Syntax:

```
int fb_open ();
```

FORTRAN Syntax:

```
SUBROUTINE FOPEN(IRET)
INTEGER*4 IRET
```

Description: This routine shall be called by the user prior to any other routine, to perform software and hardware initialization. A default environment with identifier `FBDEID` is provided.

FB_CLOSE Close a FASTBUS session.

C Syntax:

```
int fb_close ();
```

FORTRAN Syntax:

```
SUBROUTINE FCLOSE(IRET)
INTEGER*4 IRET
```

Description: When use of FASTBUS is no longer required, the user shall call this routine.

FCIENV Create an immediate execution FASTBUS environment.

C Syntax:

```
int fcienv (id_ptr);
int *id_ptr
```

FORTRAN Syntax:

```
SUBROUTINE FCIENV(IRET,ID)
INTEGER*4 IRET,ID
```

Description: Creates an immediate execution FASTBUS environment and set it to the default value. Returns the environment identifier `id`. The maximum number of simultaneously active environments is 16.

FRLENV Release a FASTBUS environment.

C Syntax:

```
int frlenv (id);
int id;
```

FORTRAN Syntax:

```
SUBROUTINE FRLENV(IRET,ID)
INTEGER*4 IRET,ID
```

Description: Release the environment with identifier `id`.

ASTBUS environment.

```
env (id);
```

```
INE FRSENV(IRET,ID)
*4 IRET,ID
```

nvironment with identifier `id` to the default state.

STBUS environment.

```
env (id, env, FPENVS, FPENVS);
 env [FPENW];
```

```
INE FSTENV(IRET,ID,ENV,FPENVS,FPENVS)
*4 IRET,ID,ENV(FPENVW)
```

essor parameters specified by the environment number `id` are returned in the

STBUS environment.

```
env (id, env, FPENVS);
 env [FPENW];
```

```
INE FSTENV(IRET,ID,ENV,FPENVS)
*4 IRET,ID,ENV(FPENVW)
```

essor parameters specified by the environment number `id` are set accordingly

ents of the `env` array.

2. OPERATIONAL PARAMETERS

FBPINI Initialize FASTBUS operational parameters.

C Syntax:

```
int fbpini (id, par_id);
int id, par_id;
```

FORTRAN Syntax:

```
SUBROUTINE FBPINI(IRET,ID,PAR_ID)
INTEGER*4 IRET,ID,PAR_ID
```

Description: Restore the default value in the operational parameter specified by par_id. If FPALL is specified as paraameter identifier, all the parameters are reset.

FBPSET Set FASTBUS operational parameter.

C Syntax:

```
int fbpset (id, par_id, par_val);
int id, par_id, par_val;
```

FORTRAN Syntax:

```
SUBROUTINE FBPSET(IRET,ID,PAR_ID,PAR_VAL)
INTEGER*4 IRET,ID,PAR_ID,PAR_VAL
```

Description: Assigns par_val to the operational parameter specified by par_id.

FBPGET Get FASTBUS operational parameter.

C Syntax:

```
int fbpget (id, par_id, par_val_ptr);
int id, par_id, *par_val_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FBPGET(IRET,ID,PAR_ID,PAR_VAL)
INTEGER*4 IRET,ID,PAR_ID,PAR_VAL
```

Description: Reads into par_val the operational parameter specified by par_id.

The implemented operational parameters are:

- **FPARBL** Arbitration level - default value is assigned by the driver at initialisation time.
- **FPEXTH** Exit severity threshold - default value is FB_SEV_ERROR. This parameter is checked inside the fb_status_report routine only, so the program will not abort after an error if fsrpt is not called.
- **FPPRTY** Control of parity generation - default value is FB_PARITY_NONE.
- **FPNOWT** Do not wait for completion of action (only valid for block transfer actions) - default value FB_FALSE.
- **FPENVS** Size in bytes of the environment - fixed value is 60.
- **FPENVW**† Size in longwords of the environment - fixed value is 15.

- **FPPRIV**† FASTBUS privileges. This parameter can be set only if the process owner is OS-9 Super User (i.e. group-user 0,0). Valid privileges are:
  - `BUSRST`: may issue a FASTBUS reset signal
  - `SRVCON`: may connect to SR interrupts
- **FPWTT**†† Wait time-out. Possible values are `FB_WTT_SHORT` (~10 ms) and `FB_WTT_LONG` (~10 min.) - default is `FB_WTT_SHORT`.

†† EXTENSION

†† NON-STANDARD: time-out value cannot be given in nanoseconds.

SIMPLE TRANSACTION ROUTINES

Single data word transfer

Read single word from Control Space.

ıtax:

```
int frc (id, prim_add, sec_add, FBVAR, sw_buf_ptr);
int id, prim_add, sec_add, *sw_buf_ptr;
```

RAN Syntax:

```
SUBROUTINE FRC(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

iption: Moves a 32 bit word from the Primary Address prim_add, Secondary Address sec_add to sw_buf.

C Write single word from Control Space.

ıtax:

```
int fwc (id, prim_add, sec_add, FBVAL, sw_buf);
int id, prim_add, sec_add, sw_buf;
```

RAN Syntax:

```
SUBROUTINE FWC(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

iption: Moves the 32 bit word sw_buf to the Primary Address prim_add and Secondary Address sec_add.

) Read single word from Data Space.

ntax:

```
int frd (id, prim_add, sec_add, FBVAR, sw_buf_ptr);
int id, prim_add, sec_add, *sw_buf_ptr;
```

RAN Syntax:

```
SUBROUTINE FRD(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

ription: Moves a 32 bit word from the Primary Address prim_add, Secondary Address sec_add to sw_buf.

D Write single word to Data Space.

ntax:

```
int fwd (id, prim_add, sec_add, FBVAL, sw_buf);
int id, prim_add, sec_add, sw_buf;
```

RAN Syntax:

```
SUBROUTINE FWD(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

ription: Moves the 32 bit word sw_buf to the Primary Address prim_add and Secondary Address sec_add.

FRCM Read single word from Control Space Multi-listener.

C Syntax:

```
int frcm (id, prim_add, sec_add, FBVAR, sw_buf_ptr);
int id, prim_add, sec_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRCM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves a 32 bit word from the Primary Address `prim_add`, Secondary Address `sec_add` to `sw_buf`.

FWCM Write single word to Control Space Multi-listener.

C Syntax:

```
int fwcm (id, prim_add, sec_add, FBVAL, sw_buf);
int id, prim_add, sec_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWCM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves the 32 bit word `sw_buf` to the Primary Address `prim_add` and Secondary Address `sec_add`.

FRDM Read single word from Data Space Multi-listener.

C Syntax:

```
int frdm (id, prim_add, sec_add, FBVAR, sw_buf_ptr);
int id, prim_add, sec_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRDM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves a 32 bit word from the Primary Address `prim_add`, Secondary Address `sec_add` to `sw_buf`.

FWDM Write single word to Data Space Multi-listener.

C Syntax:

```
int fwdm (id, prim_add, sec_add, FBVAL, sw_buf);
int id, prim_add, sec_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWDM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves the 32 bit word `sw_buf` to the Primary Address `prim_add` and Secondary Address `sec_add`.

```
:, buffer_ptr,
:, byte_count;
_ADD,FBVAR,BUFFER,
UFFER,BYTE_COUNT
```

dress prim_add, Secondary

```
:, buffer_ptr,
', byte_count;
_ADD,FBVAR,BUFFER,
UFFER,BYTE_COUNT
```

y buffer to Primary Address

```
:, buffer_ptr,
:, byte_count;
_ADD,FBVAR,BUFFER,
UFFER,BYTE_COUNT
```

dress prim_add, Secondary

FBVAR, buffer_ptr,

er_ptr, byte_count;

D,SEC_ADD,FBVAR,BUFFER,

ADD,@BUFFER,BYTE_COUNT

ointed by buffer to Primary Address

Multi-listener.

, FBVAR, buffer_ptr,

er_ptr, byte_count;

DD,SEC_ADD,FBVAR,BUFFER,

ADD,@BUFFER,BYTE_COUNT

ary Address prim_add, Secondary

fer.

ulti-listener.

, FBVAR, buffer_ptr,

er_ptr, byte_count;

DD,SEC_ADD,FBVAR,BUFFER,

ADD,@BUFFER,BYTE_COUNT

ointed by buffer to Primary Address

lti-listener.

, FBVAR, buffer_ptr,

er_ptr, byte_count;

DD,SEC_ADD,FBVAR,BUFFER,

ADD,@BUFFER,BYTE_COUNT

nary Address prim_add, Secondary

fer.

FWDBM

Block transfer write to Data Space, Multi-listener.

C Syntax:

```
int fwdbm (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FWDBM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers byte_count bytes from the array pointed by buffer to Primary Address prim_add, Secondary Address sec_add.

FIRDB†

Indirect block transfer read from Data Space.

C Syntax:

```
int firdb (id, prim_add, sec_add, FBVAR, buffer_ptr,
          max_count);
int id, prim_add, sec_add, *buffer_ptr, max_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FIRDB(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

Description: A single word read from Primary Address prim_add, Secondary Address sec_add is perfomed: the least value between this word and max_count (if greater than 0) will be used as byte counter for the block transfer. Then a single word read from Secondary Address sec_add+1 is perfomed: this value will be used as Secondary Address for the block transfer. A single word write to Secondary Address sec_add+2 and data -1 is then performed signaling the slave that the transfer is about to start. Finally a block transfer read from Data Space is performed. The word at Secondary Address sec_add+3 is reserved and should not be used.

† EXTENSION

rol Space.

```
dd, FBVAR, sw_buf_ptr);
_buf_ptr;
```

```
,ID,PRIM_ADD,FBVAR,SW_BUF)
M_ADD,SW_BUF
```

ister at Primary Address prim_add.

trol Space.

```
dd, FBVAL, sw_buf);
buf;
```

```
,ID,PRIM_ADD,FBVAR,SW_BUF)
M_ADD,SW_BUF
```

rimary Address prim_add the 32 bit word sw_buf.

Space.

```
dd, FBVAR, sw_buf_ptr);
_buf_ptr;
```

```
,ID,PRIM_ADD,FBVAR,SW_BUF)
M_ADD,SW_BUF
```

ister at Primary Address prim_add.

Space.

```
dd, FBVAL, sw_buf);
buf;
```

```
,ID,PRIM_ADD,FBVAR,SW_BUF)
M_ADD,SW_BUF
```

rimary Address prim_add the 32 bit word sw_buf.

3.4 Read length of last data transfer

FRLEN Read length of last block transfer.

C Syntax:

```
int frlen (id, FBVAR, len_ptr);
int id, *len_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRLEN(IRET,ID,PRIM_ADD,FBVAR,LEN)
INTEGER*4 ID,IRET,PRIM_ADD,LEN
```

Description: Reads from the coprocessor and stores in `len` the number of bytes transferred during the last block transfer action.

Notes: This routine returns the correct value only when no errors have occurred (this is useful for the `FIRDB` routine), or when an asynchronous error has occurred (i.e. `FEBSS2`): in all other cases the value returned is the byte counter specified by the user in the last block transfer routine, while it should return zero.

4. COMPOUND TRANSACTION ROUTINES

4.1 Access Segment Interconnect Route Table

FWRT Write SI Route Table.

C Syntax:

```
int fwrt (id, prim_add, rt_add, FBVAL, sw_buf);
int id, prim_add, rt_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWRT(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,RT_ADD,SW_BUF
```

Description: Writes the `sw_buf` entry in the SI Route Table. `prim_add` is the Primary Address of the SI, `rt_add` is the index in the route table.

FRRT Read SI Route Table.

C Syntax:

```
int frrt (id, prim_add, rt_add, FBVAR, sw_buf_ptr);
int id, prim_add, rt_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRRT(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,RT_ADD,SW_BUF
```

Description: Stores into `sw_buf` the entry indexed by `rt_add` in the SI at Primary Address `prim_add`.

FWRTB Block transfer write to SI Route Table.

C Syntax:

```
int fwrtb (id, prim_add, rt_add, FBVAR, buffer_ptr,
             byte_count);
int id, prim_add, rt_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FWRTB(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,BUFFER,
1                     BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,RT_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers `byte_count/4` entries from `buffer` to the SI at primary address `prim_add`, starting at the Route Table index `rt_add`.

FRRTB

Block transfer read from SI Route Table.

C Syntax:

```
int frrtb (id, prim_add, rt_add, FBVAR, buffer_ptr,
           byte_count);
int id, prim_add, rt_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FRRTB(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,BUFFER,
1                     BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,RT_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers `byte_count/4` entries from the SI at primary address `prim_add`, starting at the Route Table index `rt_add`, into the array pointed by `buffer`.

4.2 Read-Modify-Write FASTBUS locations

FMODC

Modify single word in Control Space.

C Syntax:

```
int fmodc (id, prim_add, sec_add, oper_id, operand);
int id, prim_add, sec_add, oper_id, operand;
```

FORTRAN Syntax:

```
SUBROUTINE FMODC(IRET,ID,PRIM_ADD,SEC_ADD,OPER_ID,OPERAND)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,OPER_ID,OPERAND
```

Description: The logical operation specified by `oper_id` is performed between the word at Primary Address `prim_add`, Secondary Address `sec_add` and `operand`. The result is written back to the FASTBUS location.

FMODD

Modify single word in Data Space.

C Syntax:

```
int fmodd (id, prim_add, sec_add, oper_id, operand);
int id, prim_add, sec_add, oper_id, operand;
```

FORTRAN Syntax:

```
SUBROUTINE FMODD(IRET,ID,PRIM_ADD,SEC_ADD,OPER_ID,OPERAND)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADDOPER_ID,OPERAND
```

Description: The logical operation specified by `oper_id` is performed between the word at Primary Address `prim_add`, Secondary Address `sec_add` and `operand`. The result is written back to the FASTBUS location.

The implemented operator identifiers are:

- **FMSET**	Set bit postions specified by `operand`
- **FMCLR**	Clear bit postions specified by `operand`
- **FMAND**	AND data read with `operand`
- **FMOR**	OR data read with `operand`
- **FMXOR**	XOR data read with `operand`

- **FMNOT** Perform ones complement on data read

FCASC† Compare and swap single word from Control Space.

C Syntax:

```
int fcasc (id, prim_add, sec_add, data_cmp_ptr, data_upd);
int id, prim_add, sec_add, data_cmp_ptr, data_upd;
```

FORTRAN Syntax:

```
 SUBROUTINE FCASC(IRET,ID,PRIM_ADD,SEC_ADD,DATA_CMP,
1                  DATA_UPD)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,DATA_CMP,DATA_UPD
```

Description: Compares the 32 bit word at Primary Address prim_add, Secondary Address sec_add for equality with the word data_cmp. If true, substitutes it with data_upd, else stores the word read in data_cmp.

FCASD† Compare and swap single word from Data Space.

C Syntax:

```
int fcasd (id, prim_add, sec_add, data_cmp_ptr, data_upd);
int id, prim_add, sec_add, data_cmp_ptr, data_upd;
```

FORTRAN Syntax:

```
 SUBROUTINE FCASD(IRET,ID,PRIM_ADD,SEC_ADD,DATA_CMP,
1                  DATA_UPD)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,DATA_CMP,DATA_UPD
```

Description: Compares the 32 bit word at Primary Address prim_add, Secondary Address sec_add for equality with the word data_cmp. If true, substitutes it with data_upd, else stores the word read in data_cmp.

FCAS2C† Compare and swap two words from Control Space.

C Syntax:

```
int fcas2c (id, prim_add, sec_add0, data_cmp0_ptr,
        data_upd0, sec_add1, data_cmp1_ptr, data_upd1);
int id, prim_add, sec_add0, *data_cmp0_ptr, data_upd0,
        sec_add1, *data_cmp1_ptr, data_upd1;
```

FORTRAN Syntax:

```
 SUBROUTINE FCAS2C(IRET,ID,PRIM_ADD,SEC_ADD0,DATA_CMP0,
1                  DATA_UPD0,SEC_ADD1,DATA_CMP1,DATA_UPD1)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD0,DATA_CMP0,DATA_UPD0,
1          SEC_ADD1,DATA_CMP1,DATA_UPD1
```

Description: Compares the 32 bit words at Primary Address prim_add, Secondary Address sec_add0 and sec_add1 for equality with the words data_cmp0 and data_cmp1 respectively. If both tests are satisfied, writes data_upd0 and data_upd1 in the FASTBUS locations, else stores the words read in data_cmp0 and data_cmp1.

FCAS2D† Compare and swap two words from Data Space.

C Syntax:

```
int fcas2d (id, prim_add, sec_add0, data_cmp0_ptr,
        data_upd0, sec_add1, data_cmp1_ptr, data_upd1);
int id, prim_add, sec_add0, *data_cmp0_ptr, data_upd0,
        sec_add1, *data_cmp1_ptr, data_upd1;
```

FORTRAN Syntax:

```
  SUBROUTINE FCAS2D(IRET,ID,PRIM_ADD,SEC_ADD0,DATA_CMP0,
1                    DATA_UPD0,SEC_ADD1,DATA_CMP1,DATA_UPD1)
  INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD0,DATA_CMP0,DATA_UPD0,
1            SEC_ADD1,DATA_CMP1,DATA_UPD1
```

Description: Compares the 32 bit words at Primary Address prim_add, Secondary Address sec_add0 and sec_add1 for equality with the words data_cmp0 and data_cmp1 respectively. If both tests are satisfied, writes data_upd0 and data_upd1 in the FASTBUS locations, else stores the words read in data_cmp0 and data_cmp1.

† EXTENSION

4.3 Send a FASTBUS Interrupt Message

FBFIM Send a FASTBUS Interrupt Message.

C Syntax:

```
int fbfim (int id, prim_add, rec_blk, FBVAR, buffer_ptr
           byte_count);
int id, prim_add, rec_blk, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
  SUBROUTINE FBFIM(IRET,ID,PRIM_ADD,REC_BLK,FBVAR,BUFFER,
1                    BYTE_COUNT)
  INTEGER*4 IRET,ID,PRIM_ADD,REC_BLK,@BUFFER,BYTE_COUNT
```

Description: Transfers byte_count bytes from the location buffer to the Interrupt receiveir block number rec_blk at Primary Address prim_add. The low-order 4 bits of the first word are replaced by byte_count/4 - 1. The second word of the message is replaced by the FASTBUS address of the Master sending the message.

Notes: Requires the EB Memory Manager driver, and linking with emmlib.l.

5. SYNCHRONIZATION, SYSTEM RESOURCE AND PORT ROUTINE

FCOMWT Wait for completion of operation.

C Syntax :

```
int fcomwt (id);
int id;
```

FORTRAN Syntax:

```
SUBROUTINE FCOMWT(IRET,ID)
INTEGER*4 IRET,ID
```

Description: This routine waits for completion of the last operation associated with the environment id. If the FPNOWT parameter is set to FB_TRUE the returned error code is associated to the results of the previous operation.

FWAI† Read FASTBUS slot number.

Syntax:

```
int fwai (FB_AEB_PORT, slot_ptr);
int *slot_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FWAI(IRET,FB_AEB_PORT,SLOT)
INTEGER*4 IRET,ID,SLOT
```

Description: Sores into slot the geographical location of the station where the module is located.

FBPRST†† Issue Reset FASTBUS.

Syntax:

```
int fbprst (FB_AEB_PORT);
```

FORTRAN Syntax:

```
SUBROUTINE FBPRST(IRET,FB_AEB_PORT)
INTEGER*4 IRET,ID,SLOT
```

Description: Issue FASTBUS Reset Bus signal.

Notes: BUSRST privilege is required.

FBVERS††† Get version numbers.

† EXTENSION

†† WARNING: In a host implementation this routine should resets the device on which the FASTBUS port is attached. Here a FASTBUS Reset Bus signal is issued.

††† NOT IMPLEMENTED

6. FASTBUS SR AND INTERRUPT MESSAGE ROUTINES

FBSRC Connect routine to SR.

C Syntax:

```
int fbsrc (FB_SR_DEFAULT, FB_AEB_PORT, procSR);
int (*procSR)();
```

Description: When an SR occurs the routine `procSR` is called if the port is enabled. It is the user responsability to find and reset the SR sorurce(s). Only one user can connect to the SR interrupt.

Notes: The user routine is called as `procSR (FB_SR_DEFAULT)`. `SRVCON` privilege is required. The signal library `signals.l` is required at link time.

FBSRD Disconnect routine from SR.

C Syntax:

```
int fbsrd (FB_SR_DEFAULT, FB_AEB_PORT);
```

Description: The connection established by `fbsrc` is broken.

Notes: `SRVCON` privilege is required. The signal library `signals.l` is required at link time.

FBSREN Enable SR connections.

C Syntax:

```
int fbsren (FB_AEB_PORT);
```

Description: The port is enabled to respond to the SR signal. SR is enabled by default when the connection is made.

Notes: `SRVCON` privilege is required.

FBSRDS Disable SR connections.

C Syntax:

```
int fbsrds (FB_AEB_PORT);
```

Description: The connected routine is not called in response to the SR signal after this routine has been called.

Notes: `SRVCON` privilege is required.

FBFIRC Connect routine to FASTBUS Interrupt Message.

C Syntax:

```
int fbfirc (FB_ENV_PORT, rec_blk, flt_mask, flt_val,
                   flt_word, procFIR);
int rec_blk, flt_mask, flt_val, flt_word, (*procFIR)();
```

FORTRAN Syntax:

```
 SUBROUTINE FBFIRC(IRET,FB_ENV_PORT,REC_BLK,FLT_MASK,
1                  FLT_VAL,PROCFIR)
 INTEGER*4 IRET,REC_BLK,FLT_MASK,FLT_VAL,PROCFIR
```

Description: When a FASTBUS Interrupt Message is detected by the receiver block number rec_blk the contents of the flt_word word of the interrupt message is ANDed with flt_mask and the result compared with flt_val. If the two are equal the routine procFIR is called, otherwise no further action is taken. Only one connection per user is allowed, and different users can connect only to different receiver block numbers.

Notes: The user routine is called as

```
procFIR (rec_blk, mess_buff, mess_len, FB_ENV_PORT)
int rec_blk, *mess_buff [16], mess_len,
```

if the connection has been done from C, and.

```
PROCFIR(REC_BLK,MESS_BUFF,MESS_LEN,FB_ENV_PORT)
INTEGER*4 REC_BLK,MESS_BUFF(16),MESS_LEN
```

when done from FORTRAN. The signal library signals.l is required at link time.

FBFIRD Disconnect routine from FASTBUS Interrupt Message.

C Syntax:

```
int fbfird (FB_ENV_PORT, rec_blk, flt_mask, flt_val,
                   flt_word, procFIR);
int rec_blk, flt_mask, flt_val, flt_word, (*procFIR)();
```

FORTRAN Syntax:

```
 SUBROUTINE FBFIRD(IRET,FB_ENV_PORT,REC_BLK,FLT_MASK,
1                  FLT_VAL,PROCFIR)
 INTEGER*4 IRET,REC_BLK,FLT_MASK,FLT_VAL,PROCFIR
```

Description: The connection established by fbfirc is broken. As only one connection per user is allowed, only the receiver block number parameter rec_blk is used by this routine.

Notes: The signal library signals.l is required at link time.

FBFIRE Enable FIR connections.

C Syntax:

```
int fbfire (FB_ENV_PORT);
```

FORTRAN Syntax:

```
SUBROUTINE FBFIRE(IRET,FB_ENV_PORT)
INTEGER*4 IRET,FB_ENV_PORT
```

Description: The receiver block specified in the connection routine is enabled to receive FASTBUS Interrupt Messages. FIR is enabled by default when the connection is made.

FBFIRS Disable FIR connections.

C Syntax:

```
int fbfirs (FB_ENV_PORT);
```

FORTRAN Syntax:

```
SUBROUTINE FBFIRS(IRET,FB_ENV_PORT)
INTEGER*4 IRET,FB_ENV_PORT
```

Description: The connected routine is not called in response to FASTBUS Interrupt Messages after this routine has been called.

nation.

```
par_hdl, wh_occ_hdl);

ass_par_hdl;
_hdl;

,IRET,ASS_PAR,WH_OCC)
SS_PAR,@WH_OCC
```

ds further status information about the last error

eturns in `ass_par` and `wh_occ` the pointers to

```
ar_ptr, wh_occ_ptr);

ss_par_ptr;
ptr;

IRET,ASS_PAR,WH_OCC)
SS_PAR,@WH_OCC
```

plays the information contained in the `ass_par`

eturns always FENORM.

efined as follows

```
    byte_counter;
}_associated_parameter;

edef struct {
har *routine_name;
nt  pc_at_exception;
B_where_occurred;
```

parameter.type can assume the following values:

d by the software

d by a coprocessor instruction

d during a FASTBUS action

d during a block transfer.

om cp_status are meaningful only if type is 1 or greater. primary_address ry_address are meaningful only if type is 2 or 3. address_register and er are meaningful only if type is 3.

occurred structure pc_at_exception is meaningful only if type is 1 or greater.

NTED

D: The standard types for the associated_parameter and where_occurred parameters are 32 bit integer values.

ERROR CODES

ie following standard error codes are defined:

EACON	FEAKTO	FEAPE	FEASS1	FEASS2	FEASS3	FEASS4
EASS5	FEASS6	FEASS7	FEBSS2	FEBUF	FECLSD	FECON
EDCON	FEDKTO	FEDPE	FEDSS1	FEDSS2	FEDSS3	FEDSS4
EDSS5	FEDSS6	FEDSS7	FEEIOV	FEENIN	FEEREL	FEERR
EFTL	FEGKTO	FEINEI	FEIPRV	FENCON	FENORM	FENPRV
EOOPS	FEOPEN	FESAPE	FESATO	FESSS1	FESSS2	FESSS3
ESSS4	FESSS5	FESSS6	FESSS7	FEUPAR	FEWTTO	

ie following standard errors codes have a special meaning:

ERR: EB Memory Manager cannot allocate space (returned by FBFIM).

FTL: FASTBUS driver not installed or incompatible with the library software version.

OOPS: unknown (or simply unimplemented) error code. On occurrence, please return us the log file with the informations displayed by fsrpt.

addition these new codes have been introduced:

ENIN Severity: **FSERR**

Environment not initialized. This error can be returned by the hardware if library calls are bypassed with direct assembler instructions. It should never occur with a proper use of the library.

APE Severity: **FSERR**

On a FASTBUS primary address cycle a parity error was encountered.

SAPE Severity: **FSERR**

On a FASTBUS secondary address cycle a parity error was encountered.

GKTO Severity: **FSERR**

GK(u) did not occurred after AG(d) within the timeout period.

short name)

```
fb_create_immediate_environment
OSE     FB_CLOSE
MWT     FB_COMPLETION_WAIT
mwt     fb_completion_wait
CON
KTO
PE
SS1
SS2
SS3
SS4
SS5
SS6
SS7
SS2
UF
LSD
ON
CON
KTO
PE
SS1
SS2
SS3
SS4
SS5
SS6
SS7
IOV
NIN
REL
RR
TL
KTO
NEI
PRV
CON
ORM
PRV
OPS
PEN
APE
ATO
SS1
SS2
SS3
SS4
SS5
```

```
FATAL
INFO
SHORT
JS_REPORT
us_report
SUCCESS
ENVIRONMENT
environment
WARNING

E_AM_I
e_am_i
E_CSR
e_csr
E_CSR_BLOCK
e_csr_block
E_CSR_BLOCK_MULT
e_csr_block_mult
E_CSR_MULT
e_csr_mult
E_CSR_SA
e_csr_sa
E_DAT
e_dat
E_DAT_BLOCK
e_dat_block
E_DAT_BLOCK_MULT
e_dat_block_mult
E_DAT_MULT
e_dat_mult
E_DAT_SA
e_datr_sa
E_ROUTE_TABLE
e_route_table

TABLE_BLOCK

table_block
```

TO THE FASTBUS DRIVER

to the FASTBUS driver

ess mode

ter to pathname string '/fastbus'

ary version number (must be $0002xxxx)

number

ated past the pathname

y bit set

error code

the FASTBUS driver

number

ry bit set

9 error code

e

US initialisation data module

number

ction code (704)

ry bit set

9 error code

Create FASTBUS environment

Function	Create FASTBUS environment and set default control word	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (768)
Output	d1.1	Event number (used for waiting end of DMA)
	d2.1	Environment number (-1 if overflow)
	d3.1	Default Control word
	(a0)	Pointer to environment space in the static storage
Error	cc	Carry bit set
	d0.1	FASTBUS error code

Release FASTBUS environment

Function	Release FASTBUS environment	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (769)
	d2.1	Environment number (-1 releases all env's owned by the process)
Output	none	
Error	cc	Carry bit set
	d0.1	FASTBUS error code

Reset FASTBUS environment

Function	Reset Environment (disconnect all and reset Control word)	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (770)
	d2.1	Environment number
Output	d3.1	Default Control word
Error	cc	Carry bit set
	d0.1	FASTBUS error code

Set Control word

Function	Set Environment control word (and check for DMA enabled)	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (771)
	d2.1	Environment number
	d3.1	Control word
Output	none	
Error	cc	Carry bit set
	d0.1	FASTBUS error code

Set privileges

Function	Check SETPRV privilege and set privileges	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (772)
	d2.1	Environment number
	d3.1	Required privileges mask
Output	none	
Error	cc	Carry bit set
	d0.1	FASTBUS error code

Connect SR

Function	Connect or disconnect Service Request	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (800)
	d2.l	Environment number
	d3.l	0 to connect / -1 to disconnect
	d5.l	Signal number to be used
Output	none	
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Connect FIR

Function	Connect or disconnect FASTBUS Interrupt Message	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (801)
	d2.l	Environment number
	d3.l	0 to connect / -1 to disconnect
	d4.l	Receiver block number
	d5.l	Signal number to be used
Output	none	
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Enable SR

Function	Enable or disable Service Request	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (802)
	d2.l	Environment number
	d3.l	0 to disable / 1 to enable
Output	none	
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Enable FIR

Function	Enable or disable FASTBUS Interrupt Message	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (803)
	d2.l	Environment number
	d3.l	0 to disable / 1 to enable
Output	none	
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Get Info

Function	Get pointer to static storage information (used by `fbmon`)	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (832)
Output	(a0)	Pointer to info structure
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Get Error name

Function	Get pointer to FASTBUS error name	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (833)
	d4.l	FASTBUS error code
Output	(a0)	Pointer to error string
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Get Instruction name

Function	Get pointer to coprocessor instruction name	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (834)
	d4.l	Coprocessor instruction code
Output	(a0)	Pointer to instruction string
Error	cc	Carry bit set
	d0.l	FASTBUS error code

est.c

```
RETURN CODE */



_o);   /* GET INFO */
       /* REPORT ERROR */



       /* OPEN SESSION */

6);    /* SET ARBITRATION LEVEL */

NVS, FPENVS);

us word = $%x \n", *env);

);     /* FIND SLOT */

d \n", slot);

eration:\n");

;

FBVAR, &csr0);  /* READ CSR */

);
       /* CLOSE SESSION */
```

(Version 2.0 of 88-62)
(Version 2.0 of 88-12)

ALEPH 88-146
DATACQ 88-21
A. Miotto et al.
14.11.1988

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

Aleph Event Builder
FASTBUS library v2.0

A.Castro, A. Miotto.

A library has been written to allow multi-user utilization of FASTBUS standard routines on the Aleph Event Builder under the OS-9/68K operating system.

Authors: A.Castro, A. Miotto.
Network Address: AL0VOL::MIOTTO at CERN
Version of Document: 2.0
Revision date: 3 November 1988
Status: First update

Changes from last version:

introduction

- new paragraphs:
  THE DRIVER
  THE FBINIT MODULE
  THE FASTBUS MONITOR
  CHANGES FROM PREVIOUS RELEASE

FASTBUS routines

- new routines: FBPINI, FRLEN, FMODC, FMODD, FBFIM, FRRTB, FWRTB.
- FCASC, FCASD syntax changed.
- FCASC2 and FCASD2 modified to FCAS2C and FCAS2D (syntax changed).
- FPNOWT parameter default set to FFALSE.
- FPWTT parameter added.

Appendix

- low-level call documentation added

llows FASTBUS actions to be
multaneous users and most of

of the coprocessor under the
ing, and is called by the library
ient management and interrupt
o avoid software overhead: the
ions.

standard defined by U.S. NIM
of the coprocessor and to keep

N and C languges. Calling
by value; the FASTBUS error
neters passed by reference.

lowing commands should be executed (for example from a

```
! load driver in memory
! load descriptor in memory
! initialize it
```

must be installed in order to work properly.

am must contain a call to `fb_open` before any other call

fier `FB_DEFAULT_EID` is created and initialised.

a maximum of 16 FASTBUS environments.

should always be examined by the user. In case of values

output available information on the standard error output

se should be called before exiting.

owed. For this reason FORTRAN entry points for SR

ired to make the connection.

ion per task, and one connection per receiver block number

ect to the same block).

S names considered in this implementation are defined in

her names are defined only in the short form, excepted the

n incompatibility would have arosen with the standard C

long names `fb_open` and `fb_close` are used instead.

.

include file `cfbdef.h` should be placed in the `DEFS`

b.l in the `LIB` subdirectory. If the `make` utility is loaded,

dard C libraries can be compiled with the command:

```
! without the .c extension
```

ple program showing the use of several FASTBUS calls

ual.

ndler takes care of releasing environments if the user does

turely aborted. With OS-9 V2.1 some situations arose in

the system; this seems to have been fixed with OS-9 V2.2.

hat all process using FASTBUS are stopped and then type:

```
niz fastbus
z fastbus
```

RENCES FROM THE STANDARD Any difference from the Standard FASTBUS software with one ore more † symbols in the following. These conditions can be met:

MPLEMENTED means a category "A" (mandatory) routine that has not been implemented. The only at could not be implemented without avoiding inacceptable overheads is `fsgsum` (decode summary ther missing routines will be implemented in following releases.

NSION means a routine or a parameter not defined in the standard and meaningful only in this tation.

STANDARD means that the specified routine or parameter has been modified from the standard s.

RIVER The description of low-level calls to the FASTBUS driver is in Appendix B. It is added eteness but use of direct calls is not reccomended.

Binit DATA MODULE Some parameters have to be provided to initialise the driver. They arbitration level, needed to avoid conflicts between Masters on the same create; – the FASTBUS the EB, used when sending Interrupt Messages. They should be stored in a data module named If the module is not found, an arbitration level of 31 and a null primary address are used.

`fbin` utility can be used to build this module. If you run interactively, it asks for the parameters to ced. Alternatively, you can run it from the private startup file of your EB:

```
n >/nil                      !no output
tration_level>
nary_address>
```

FASTBUS MONITOR The command `moni -f` is no longer valid, starting from release 2.0. on monitor shows the total and per process value of exception and interrupt counters, plus some rmation related with each environment.

NGES FROM RELEASE 1.1 Some new instructions have been introduced in the coprocessor e since the last release. Two of them are used for accessing data in the coprocessor itself, and permit a ndling of environments both in the library and in the driver. Unfortunately the old library is not ly compatible with the new driver, because some of the low level calls have changed, and this could nge behaviour in old code (like sleeping forever in programs waiting for end-of-DMA interrupts). For vill receive a FEFTL error and will

n 4) are installed in your EB, you will
lated include file cfbdef.h. No such

lease note that the coprocessor does not
hronous part of the instruction, and that
following fragment of code shows a

```
,sec, FBVAR, buffer, count);
);
) actual_count = 0;
```

ementary and reading the field
Here the correct value is stored on any
tent is meaningless. In any case
th of data transferred using a firdb

ASTBUS word;
itine requires the EB Memory Manger
h the emmlib.l library;
able_block: these routines do not
em by executing a loop of frrt and

swapped value were not rturned to the

asd2) the syntax has changed;
als.l;
nect (except the message address) are
by reference if the FORTRAN one has

as defined in the standard. It is anyway
tely to the desired value;
duced. The time-out value cannot be
FB_WTT_SHORT (~10 ms, default) and

[1] U.S. NIM Committee - FASTBUS standard routines - March 1987 DOE/ER-0325

FASTBUS ROUTINES

)NMENT MANAGEMENT

Open a FASTBUS session.

```
int fb_open ();
```

```
x: SUBROUTINE FOPEN(IRET)
   INTEGER*4 IRET
```

This routine shall be called by the user prior to any other routine, to perform software and hardware initialization. A default environment with identifier FBDEID is provided.

Close a FASTBUS session.

```
int fb_close ();
```

```
x: SUBROUTINE FCLOSE(IRET)
   INTEGER*4 IRET
```

When use of FASTBUS is no longer required, the user shall call this routine.

Create an immediate execution FASTBUS environment.

```
int fcienv (id_ptr);
int *id_ptr
```

```
x: SUBROUTINE FCIENV(IRET,ID)
   INTEGER*4 IRET,ID
```

Creates an immediate execution FASTBUS environment and set it to the default value. Returns the environment identifier id. The maximum number of simultaneously active environments is 16.

Release a FASTBUS environment.

```
int frlenv (id);
int id;
```

```
x: SUBROUTINE FRLENV(IRET,ID)
   INTEGER*4 IRET,ID
```

Release the environment with identifier id.

: state.

. FPENVS)

ent number `id` are returned in the

ent number `id` are set accordingly

2. OPERATIONAL PARAMETERS

FBPINI Initialize FASTBUS operational parameters.

C Syntax:

```
int fbpini (id, par_id);
int id, par_id;
```

FORTRAN Syntax:

```
SUBROUTINE FBPINI(IRET,ID,PAR_ID)
INTEGER*4 IRET,ID,PAR_ID
```

Description: Restore the default value in the operational parameter specified by `par_id`. If `FPALL` is specified as paraameter identifier, all the parameters are reset.

FBPSET Set FASTBUS operational parameter.

C Syntax:

```
int fbpset (id, par_id, par_val);
int id, par_id, par_val;
```

FORTRAN Syntax:

```
SUBROUTINE FBPSET(IRET,ID,PAR_ID,PAR_VAL)
INTEGER*4 IRET,ID,PAR_ID,PAR_VAL
```

Description: Assigns `par_val` to the operational parameter specified by `par_id`.

FBPGET Get FASTBUS operational parameter.

C Syntax:

```
int fbpget (id, par_id, par_val_ptr);
int id, par_id, *par_val_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FBPGET(IRET,ID,PAR_ID,PAR_VAL)
INTEGER*4 IRET,ID,PAR_ID,PAR_VAL
```

Description: Reads into `par_val` the operational parameter specified by `par_id`.

The implemented operational parameters are:

- **FPARBL** Arbitration level - default value is assigned by the driver at initialisation time.
- **FPEXTH** Exit severity threshold - default value is `FB_SEV_ERROR`. This parameter is checked inside the `fb_status_report` routine only, so the program will not abort after an error if `fsrpt` is not called.
- **FPPRTY** Control of parity generation - default value is `FB_PARITY_NONE`.
- **FPNOWT** Do not wait for completion of action (only valid for block transfer actions) - default value `FB_FALSE`.
- **FPENVS** Size in bytes of the environment - fixed value is `60`.
- **FPENVW**[†] Size in longwords of the environment - fixed value is `15`.

- **`FPPRIV`**[†] FASTBUS privileges. This parameter can be set only if the process owner is OS-9 Super User (i.e. group-user 0,0). Valid privileges are:
  - `BUSRST`: may issue a FASTBUS reset signal
  - `SRVCON`: may connect to SR interrupts
- **`FPWTT`**[††] Wait time-out. Possible values are `FB_WTT_SHORT` (~10 ms) and `FB_WTT_LONG` (~10 min.) - default is `FB_WTT_SHORT`.

[††] EXTENSION

[††] NON-STANDARD: time-out value cannot be given in nanoseconds.

INES

ontrol Space.

```
d, sec_add, FBVAR, sw_buf_ptr);
c_add, *sw_buf_ptr;
```

```
ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
IM_ADD,SEC_ADD,SW_BUF
```

he Primary Address prim_add, Secondary Address

ontrol Space.

```
d, sec_add, FBVAL, sw_buf);
c_add, sw_buf;
```

```
ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
IM_ADD,SEC_ADD,SW_BUF
```

uf to the Primary Address prim_add and Secondary

ata Space.

```
d, sec_add, FBVAR, sw_buf_ptr);
c_add, *sw_buf_ptr;
```

```
ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
IM_ADD,SEC_ADD,SW_BUF
```

he Primary Address prim_add, Secondary Address

Space.

```
d, sec_add, FBVAL, sw_buf);
c_add, sw_buf;
```

```
ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
IM_ADD,SEC_ADD,SW_BUF
```

uf to the Primary Address prim_add and Secondary

FRCM Read single word from Control Space Multi-listener.

C Syntax:

```
int frcm (id, prim_add, sec_add, FBVAR, sw_buf_ptr);
int id, prim_add, sec_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRCM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves a 32 bit word from the Primary Address prim_add, Secondary Address sec_add to sw_buf.

FWCM Write single word to Control Space Multi-listener.

C Syntax:

```
int fwcm (id, prim_add, sec_add, FBVAL, sw_buf);
int id, prim_add, sec_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWCM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves the 32 bit word sw_buf to the Primary Address prim_add and Secondary Address sec_add.

FRDM Read single word from Data Space Multi-listener.

C Syntax:

```
int frdm (id, prim_add, sec_add, FBVAR, sw_buf_ptr);
int id, prim_add, sec_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRDM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves a 32 bit word from the Primary Address prim_add, Secondary Address sec_add to sw_buf.

FWDM Write single word to Data Space Multi-listener.

C Syntax:

```
int fwdm (id, prim_add, sec_add, FBVAL, sw_buf);
int id, prim_add, sec_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWDM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,SW_BUF)
INTEGER*4 IRET,ID,PRIM_ADD,SEC_ADD,SW_BUF
```

Description: Moves the 32 bit word sw_buf to the Primary Address prim_add and Secondary Address sec_add.

3.2 Block transfers

FRCB Block transfer read from Control Space.

C Syntax:

```
int frcb (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FRCB(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                   BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers byte_count bytes from the Primary Address prim_add, Secondary Address sec_add, to the array pointed by buffer.

FWCB Block transfer write to Control Space.

C Syntax:

```
int fwcb (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FWCB(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                   BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers byte_count bytes from the array pointed by buffer to Primary Address prim_add, Secondary Address sec_add.

FRDB Block transfer read from Data Space.

C Syntax:

```
int frdb (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FRDB(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                   BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers byte_count bytes from the Primary Address prim_add, Secondary Address sec_add, to the array pointed by buffer.

Block transfer write to Data Space.

```
int fwdb (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

Syntax:

```
 SUBROUTINE FWDB(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                 BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

: Transfers byte_count bytes from the array pointed by buffer to Primary Address prim_add, Secondary Address sec_add.

Block transfer read from Control Space, Multi-listener.

```
int frcbm (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

Syntax:

```
 SUBROUTINE FRCBM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                 BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

: Transfers byte_count bytes from the Primary Address prim_add, Secondary Address sec_add, to the array pointed by buffer.

Block transfer write to Control Space, Multi-listener.

```
int fwcbm (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

Syntax:

```
 SUBROUTINE FWCBM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                 BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

: Transfers byte_count bytes from the array pointed by buffer to Primary Address prim_add, Secondary Address sec_add.

Block transfer read from Data Space, Multi-listener.

```
int frdbm (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

Syntax:

```
 SUBROUTINE FRDBM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                 BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

: Transfers byte_count bytes from the Primary Address prim_add, Secondary Address sec_add, to the array pointed by buffer.

FWDBM Block transfer write to Data Space, Multi-listener.

C Syntax:

```
int fwdbm (id, prim_add, sec_add, FBVAR, buffer_ptr,
          byte_count);
int id, prim_add, sec_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FWDBM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                  BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers byte_count bytes from the array pointed by buffer to Primary Address prim_add, Secondary Address sec_add.

FIRDB† Indirect block transfer read from Data Space.

C Syntax:

```
int firdb (id, prim_add, sec_add, FBVAR, buffer_ptr,
          max_count);
int id, prim_add, sec_add, *buffer_ptr, max_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FIRDB(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFFER,
1                  BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,SEC_ADD,@BUFFER,BYTE_COUNT
```

Description: A single word read from Primary Address prim_add, Secondary Address sec_add is perfomed: the least value between this word and max_count (if greater than 0) will be used as byte counter for the block transfer. Then a single word read from Secondary Address sec_add+1 is perfomed: this value will be used as Secondary Address for the block transfer. A single word write to Secondary Address sec_add+2 and data -1 is then performed signaling the slave that the transfer is about to start. Finally a block transfer read from Data Space is performed. The word at Secondary Address sec_add+3 is reserved and should not be used.

† EXTENSION

3.3 Secondary address routines

FRCSA Read NTA register in Control Space.

C Syntax:

```
int frcsa (id, prim_add, FBVAR, sw_buf_ptr);
int id, prim_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRCSA(IRET,ID,PRIM_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,SW_BUF
```

Description: Stores in `sw_buf` the NTA register at Primary Address `prim_add`.

FWCSA Write NTA register in Control Space.

C Syntax:

```
int fwcsa (id, prim_add, FBVAL, sw_buf);
int id, prim_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWCSA(IRET,ID,PRIM_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,SW_BUF
```

Description: Writes in the NTA register at Primary Address `prim_add` the 32 bit word `sw_buf`.

FRDSA Read NTA register in Data Space.

C Syntax:

```
int frdsa (id, prim_add, FBVAR, sw_buf_ptr);
int id, prim_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRDSA(IRET,ID,PRIM_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,SW_BUF
```

Description: Stores in `sw_buf` the NTA register at Primary Address `prim_add`.

FWDSA Write NTA register in Data Space.

C Syntax:

```
int fwdsa (id, prim_add, FBVAL, sw_buf);
int id, prim_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWDSA(IRET,ID,PRIM_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,SW_BUF
```

Description: Writes in the NTA register at Primary Address `prim_add` the 32 bit word `sw_buf`.

data transfer

gth of last block transfer.

```
en (id, FBVAR, len_ptr);
 *len_ptr;

INE FRLEN(IRET,ID,PRIM_ADD,FBVAR,LEN)
*4 ID,IRET,PRIM_ADD,LEN
```

n the coprocessor and stores in `len` the number of bytes transferred during ck transfer action.

e returns the correct value only when no errors have occurred (this is useful RDB routine), or when an asynchronous error has occurred (i.e. `FEBSS2`): in ases the value returned is the byte counter specified by the user in the last sfer routine, while it should return zero.

4. COMPOUND TRANSACTION ROUTINES

4.1 Access Segment Interconnect Route Table

FWRT Write SI Route Table.

C Syntax:

```
int fwrt (id, prim_add, rt_add, FBVAL, sw_buf);
int id, prim_add, rt_add, sw_buf;
```

FORTRAN Syntax:

```
SUBROUTINE FWRT(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,RT_ADD,SW_BUF
```

Description: Writes the `sw_buf` entry in the SI Route Table. `prim_add` is the Primary Address of the SI, `rt_add` is the index in the route table.

FRRT Read SI Route Table.

C Syntax:

```
int frrt (id, prim_add, rt_add, FBVAR, sw_buf_ptr);
int id, prim_add, rt_add, *sw_buf_ptr;
```

FORTRAN Syntax:

```
SUBROUTINE FRRT(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,SW_BUF)
INTEGER*4 ID,IRET,PRIM_ADD,RT_ADD,SW_BUF
```

Description: Stores into `sw_buf` the entry indexed by `rt_add` in the SI at Primary Address `prim_add`.

FWRTB Block transfer write to SI Route Table.

C Syntax:

```
int fwrtb (id, prim_add, rt_add, FBVAR, buffer_ptr,
              byte_count);
int id, prim_add, rt_add, *buffer_ptr, byte_count;
```

FORTRAN Syntax:

```
 SUBROUTINE FWRTB(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,BUFFER,
1                    BYTE_COUNT)
 INTEGER*4 ID,IRET,PRIM_ADD,RT_ADD,@BUFFER,BYTE_COUNT
```

Description: Transfers `byte_count/4` entries from `buffer` to the SI at primary address `prim_add`, starting at the Route Table index `rt_add`.

```
AR, buffer_ptr,
r, byte_count;
T_ADD,FBVAR,BUFFER,
BUFFER,BYTE_COUNT
```

t primary address prim_add,

ray pointed by buffer.

```
er_id, operand);
operand;
```

```
C_ADD,OPER_ID,OPERAND)
PER_ID,OPERAND
```

med between the word at Primary

id and operand. The result is

```
er_id, operand);
operand;
```

```
C_ADD,OPER_ID,OPERAND)
ER_ID,OPERAND
```

med between the word at Primary

id and operand. The result is

ol Space.

```
ata_cmp_ptr, data_upd);
_ptr, data_upd;
```

```
SEC_ADD,DATA_CMP,
```

```
,DATA_CMP,DATA_UPD
```

›rim_add, Secondary Address

cmp. If true, substitutes it with

p.

pace.

```
ata_cmp_ptr, data_upd);
_ptr, data_upd;
```

```
SEC_ADD,DATA_CMP,
```

```
,DATA_CMP,DATA_UPD
```

›rim_add, Secondary Address

cmp. If true, substitutes it with

p.

Space.

```
 data_cmp0_ptr,
mp1_ptr, data_upd1);
mp0_ptr, data_upd0,
ata_upd1;
```

```
,SEC_ADD0,DATA_CMP0,
1,DATA_CMP1,DATA_UPD1)
0,DATA_CMP0,DATA_UPD0,
UPD1
```

prim_add, Secondary Address

h the words data_cmp0 and

isfied, writes data_upd0 and

es the words read in data_cmp0

```
ptr,
ta_upd1);
ta_upd0,

ATA_CMP0,
,DATA_UPD1)
,DATA_UPD0,
```

econdary Address

ata_cmp0 and

ata_upd0 and

l in data_cmp0

```
uffer_ptr

ount;

AR,BUFFER,

TE_COUNT
```

Interrupt receiveir

order 4 bits of the

l of the message is

.

.1.

TION, SYSTEM RESOURCE AND PORT ROUTINE

t for completion of operation.

```
 fcomwt (id);
 id;
```

```
ROUTINE FCOMWT(IRET,ID)
EGER*4 IRET,ID
```

routine waits for completion of the last operation associated with the environment

If the FPNOWT parameter is set to FB_TRUE the returned error code is associated

e results of the previous operation.

d FASTBUS slot number.

```
 fwai (FB_AEB_PORT, slot_ptr);
 *slot_ptr;
```

```
ROUTINE FWAI(IRET,FB_AEB_PORT,SLOT)
EGER*4 IRET,ID,SLOT
```

s into slot the geographical location of the station where the module is located.

ie Reset FASTBUS.

```
 fbprst (FB_AEB_PORT);
```

```
ROUTINE FBPRST(IRET,FB_AEB_PORT)
EGER*4 IRET,ID,SLOT
```

e FASTBUS Reset Bus signal.

RST privilege is required.

version numbers.

n a host implementation this routine should resets the device on which the FASTBUS ort is attached. Here a FASTBUS Reset Bus signal is issued.

D

6. FASTBUS SR AND INTERRUPT MESSAGE ROUTINES

FBSRC

Connect routine to SR.

C Syntax:

```
int fbsrc (FB_SR_DEFAULT, FB_AEB_PORT, procSR);
int (*procSR)();
```

Description: When an SR occurs the routine procSR is called if the port is enabled. It is the user responsability to find and reset the SR sorurce(s). Only one user can connect to the SR interrupt.

Notes: The user routine is called as procSR (FB_SR_DEFAULT). SRVCON privilege is required. The signal library signals.l is required at link time.

FBSRD

Disconnect routine from SR.

C Syntax:

```
int fbsrd (FB_SR_DEFAULT, FB_AEB_PORT);
```

Description: The connection established by fbsrc is broken.

Notes: SRVCON privilege is required. The signal library signals.l is required at link time.

FBSREN

Enable SR connections.

C Syntax:

```
int fbsren (FB_AEB_PORT);
```

Description: The port is enabled to respond to the SR signal. SR is enabled by default when the connection is made.

Notes: SRVCON privilege is required.

FBSRDS

Disable SR connections.

C Syntax:

```
int fbsrds (FB_AEB_PORT);
```

Description: The connected routine is not called in response to the SR signal after this routine has been called.

Notes: SRVCON privilege is required.

```
_val,
cFIR)();
_MASK,
R
```

r block number ssage is ANDed wo are equal the y one connection nt receiver block

```
_PORT)
```

```
T)
```

ired at link time.

```
_val,
cFIR)();
_MASK,
R
```

ection per user is by this routine.

ceive FASTBUS nade.

FBFIRS Disable FIR connections.

C Syntax:

```
int fbfirs (FB_ENV_PORT);
```

FORTRAN Syntax:

```
SUBROUTINE FBFIRS(IRET,FB_ENV_PORT)
INTEGER*4 IRET,FB_ENV_PORT
```

Description: The connected routine is not called in response to FASTBUS Interrupt Messages after this routine has been called.

ATUS AND ERROR HANDLING

UM† Decode summary status.

UP†† Find supplementary status information.

x:

```
int fsfsup (id, iret, ass_par_hdl, wh_occ_hdl);
int id, iret;
FB_associated_parameter **ass_par_hdl;
FB_where_occurred **wh_occ_hdl;
```

AN Syntax:

```
SUBROUTINE FSFSUP(IRET0,ID,IRET,ASS_PAR,WH_OCC)
INTEGER*4 IRET0,ID,IRET,@ASS_PAR,@WH_OCC
```

tion: To be called if `iret!=FENORM`. Finds further status information about the last error produced by a FASTBUS action, and returns in `ass_par` and `wh_occ` the pointers to the supplementary status structures.

T†† Report a FASTBUS error

x:

```
int fsrpt (id, iret, ass_par_ptr, wh_occ_ptr);
int id, iret;
FB_associated_parameter *ass_par_ptr;
FB_where_occurred *wh_occ_ptr;
```

AN Syntax:

```
SUBROUTINE FSRPT(IRET0,ID,IRET,ASS_PAR,WH_OCC)
INTEGER*4 IRET0,ID,IRET,@ASS_PAR,@WH_OCC
```

tion: To be called if `iret!=FENORM`. Displays the information contained in the `ass_par` and `wh_occ` stuctures. This routine returns always `FENORM`.

ociated_parameter and where_occurred structures are defined as follows

```
typedef struct {
  int  type,
       id,
       error_code,
       severity_level,
  char *error_name;
  int  cp_status;
  char *instr_name;
  int  primary_address,
       secondary_address,
       address_register,
counter;
ted_parameter;

t {
ine_name;
_exception;
ccurred;
```

r.type can assume the following values:

vare

cessor instruction

\STBUS action

ock transfer.

tus are meaningful only if type is 1 or greater. primary_address

ss are meaningful only if type is 2 or 3. address_register and

ingful only if type is 3.

structure pc_at_exception is meaningful only if type is 1 or greater.

dard types for the associated_parameter and where_occurred parameters
oit integer values.

ERROR CODES

- The following standard error codes are defined:

FEACON	**FEAKTO**	**FEAPE**	**FEASS1**	**FEASS2**	**FEASS3**	**FEASS4**
FEASS5	**FEASS6**	**FEASS7**	**FEBSS2**	**FEBUF**	**FECLSD**	**FECON**
FEDCON	**FEDKTO**	**FEDPE**	**FEDSS1**	**FEDSS2**	**FEDSS3**	**FEDSS4**
FEDSS5	**FEDSS6**	**FEDSS7**	**FEEIOV**	**FEENIN**	**FEEREL**	**FEERR**
FEFTL	**FEGKTO**	**FEINEI**	**FEIPRV**	**FENCON**	**FENORM**	**FENPRV**
FEOOPS	**FEOPEN**	**FESAPE**	**FESATO**	**FESSS1**	**FESSS2**	**FESSS3**
FESSS4	**FESSS5**	**FESSS6**	**FESSS7**	**FEUPAR**	**FEWTTO**	

- The following standard errors codes have a special meaning:

FEERR: EB Memory Manager cannot allocate space (returned by `FBFIM`).

FEFTL: FASTBUS driver not installed or incompatible with the library software version.

FEOOPS: unknown (or simply unimplemented) error code. On occurrence, please return us the log file with the informations displayed by `fsrpt`.

- In addition these new codes have been introduced:

FEENIN Severity: **FSERR**

Environment not initialized. This error can be returned by the hardware if library calls are bypassed with direct assembler instructions. It should never occur with a proper use of the library.

FEAPE Severity: **FSERR**

On a FASTBUS primary address cycle a parity error was encountered.

FESAPE Severity: **FSERR**

On a FASTBUS secondary address cycle a parity error was encountered.

FEGKTO Severity: **FSERR**

GK(u) did not occurred after AG(d) within the timeout period.

APPENDIX

A - LIST OF RESERVED NAMES (sorted by short name)

short name	long name
	fb_close[1]
	fb_open
FBAEBP	FB_AEB_PORT
FBDEID	FB_DEFAULT_EID
FBENVP	FB_ENV_PORT
FBFIM	FB_SEND_FIM
fbfim	fb_send_fim
FBFIRC	FB_FIR_CONNECT
fbfirc	fb_fir_connect
FBFIRD	FB_FIR_DISCONNECT
fbfird	fb_fir_disconnect
FBFIRE	FB_FIR_ENABLE
fbfire	fb_fir_enable
FBFIRS	FB_FIR_DISABLE
fbfirs	fb_fir_disable
FBINID	FB_INVALID_EID
FBPGET	FB_PAR_GET
fbpget	fb_par_get
FBPINI	FB_PAR_INIT
fbpini	fb_par_init
FBPRST	FB_PORT_RESET
fbprst	fb_port_reset
FBPSET	FB_PAR_SET
fbpset	fb_par_set
fbsrc	fb_sr_connect
fbsrd	fb_sr_disconnect
FBSRDF	FB_SR_DEFAULT
fbsrds	fb_sr_disable
fbsren	fb_sr_enable
FBVAL	FB_BUFFER_VAL
FBVAR	FB_BUFFER_VAR
FCASC	FB_CAS_CSR
fcasc	fb_cas_csr
FCAS2C	FB_CAS2_CSR
fcas2c	fb_cas2_csr
FCASD	FB_CAS_DAT
fcasd	fb_cas_dat
FCAS2D	FB_CAS2_DAT
fcas2d	fb_cas2_dat
FCIENV	FB_CREATE_IMMEDIATE_ENVIRONMENT
fcienv	fb_create_immediate_environment
FCLOSE	FB_CLOSE
FCOMWT	FB_COMPLETION_WAIT
fcomwt	fb_completion_wait
FEACON	
FEAKTO	
FEAPE	
FEASS1	
FEASS2	
FEASS3	
FEASS4	
FEASS5	
FEASS6	
FEASS7	
FEBSS2	
FEBUF	
FECLSD	
FECON	
FEDCON	
FEDKTO	
FEDPE	
FEDSS1	
FEDSS2	
FEDSS3	
FEDSS4	
FEDSS5	
FEDSS6	
FEDSS7	
FEEIOV	
FEENIN	
FEEREL	
FEERR	
FEFTL	
FEGKTO	
FEINEI	
FEIPRV	
FENCON	
FENORM	
FENPRV	
FEOOPS	
FEOPEN	
FESAPE	
FESATO	
FESSS1	
FESSS2	
FESSS3	
FESSS4	
FESSS5	

[1] Lowercase names indicate C entry points, while the same name in uppercase are used for FORTRAN entry points.

	_FALSE
	_GET_ENVIRONMENT
	_get_environment
	_IND_READ_DAT_BLOCK
	_ind_read_dat_block
	_WTT_LONG
	_MODIFY_CSR
	_modify_csr
	_MODIFY_DAT
	_modify_dat
	_OPEN
	_PARITY_EVEN
	_PARITY_NONE
	_PARITY_ODD
	_READ_CSR
	_read_csr
	_READ_CSR_BLOCK
	_read_csr_block
	_READ_CSR_BLOCK_MULT
	_read_csr_block_mult
	_READ_CSR_MULT
	_read_csr_mult
	_READ_CSR_SA
	_read_csr_sa
	_READ_DAT
	_read_dat
	_READ_DAT_BLOCK
	_read_dat_block
	_READ_DAT_BLOCK_MULT
	_read_dat_block_mult
	_READ_DAT_MULT
	_read_dat_mult
	_READ_DAT_SA
	_read_dat_sa
	_READ_LENGTH
	_read_length
	_RELEASE_ENVIRONMENT
	_release_environment
	_READ_ROUTE_TABLE
	_read_route_table
	OUTE_TABLE_BLOCK
	oute_table_block
	_RESET_ENVIRONMENT
	_reset_environment
	_SEV_ERROR
	_FIND_SUPPLEMENTARY
	_find_supplementary

FSFTL	FB_SEV_FATAL
FSINFO	FB_SEV_INFO
FSHORT	FB_WTT_SHORT
FSRPT	FB_STATUS_REPORT
fsrpt	fb_status_report
FSSUCC	FB_SEV_SUCCESS
FSTENV	FB_SET_ENVIRONMENT
fstenv	fb_set_environment
FSWARN	FB_SEV_WARNING
FTRUE	FB_TRUE
FWAI	FB_WHERE_AM_I
fwai	fb_where_am_i
FWC	FB_WRITE_CSR
fwc	fb_write_csr
FWCB	FB_WRITE_CSR_BLOCK
fwcb	fb_write_csr_block
FWCBM	FB_WRITE_CSR_BLOCK_MULT
fwcbm	fb_write_csr_block_mult
FWCM	FB_WRITE_CSR_MULT
fwcm	fb_write_csr_mult
FWCSA	FB_WRITE_CSR_SA
fwcsa	fb_write_csr_sa
FWD	FB_WRITE_DAT
fwd	fb_write_dat
FWDB	FB_WRITE_DAT_BLOCK
fwdb	fb_write_dat_block
FWDBM	FB_WRITE_DAT_BLOCK_MULT
fwdbm	fb_write_dat_block_mult
FWDM	FB_WRITE_DAT_MULT
fwdm	fb_write_dat_mult
FWDSA	FB_WRITE_DAT_SA
fwdsa	fb_write_datr_sa
FWRT	FB_WRITE_ROUTE_TABLE
fwrt	fb_write_route_table
FWRTB	FB_WRITE_ROUTE_TABLE_BLOCK
fwrtb	fb_write_route_table_block

CALLS TO THE FASTBUS DRIVER

driver

pen a path to the FASTBUS driver

Open

).b Access mode

)) Pointer to pathname string '/fastbus'

.l Library version number (must be $0002xxxx)

.w Path number

)) Updated past the pathname

Carry bit set

.w Os-9 error code

driver

ose path to the FASTBUS driver

Close

).b Path number

ne

Carry bit set

l.w Os-9 error code

ta module

ead FASTBUS initialisation data module

GetStt

).b Path number

1.w Function code (704)

one

Carry bit set

1.w Os-9 error code

nvironment

te FASTBUS environment and set default control word

tStt

Path number

v Function code (768)

Event number (used for waiting end of DMA)

Environment number (-1 if overflow)

Default Control word

Pointer to environment space in the static storage

Carry bit set

FASTBUS error code

environment

ase FASTBUS environment

tStt

Path number

v Function code (769)

Environment number (-1 releases all env's owned by the process)

Carry bit set

FASTBUS error code

nvironment

t Environment (disconnect all and reset Control word)

etStt

Path number

w Function code (770)

Environment number

Default Control word

Carry bit set

FASTBUS error code

Set Control word

Function	Set Environment control word (and check for DMA enabled)	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (771)
	d2.l	Environment number
	d3.l	Control word
Output	none	
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Set privileges

Function	Check SETPRV privilege and set privileges	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (772)
	d2.l	Environment number
	d3.l	Required privileges mask
Output	none	
Error	cc	Carry bit set
	d0.l	FASTBUS error code

Connect SR

Function	Connect or disconnect Service Request	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (800)
	d2.1	Environment number
	d3.1	0 to connect / -1 to disconnect
	d5.1	Signal number to be used
Output	none	
Error	cc	Carry bit set
	d0.1	FASTBUS error code

Connect FIR

Function	Connect or disconnect FASTBUS Interrupt Message	
System call	I$GetStt	
Input	d0.b	Path number
	d1.w	Function code (801)
	d2.1	Environment number
	d3.1	0 to connect / -1 to disconnect
	d4.1	Receiver block number
	d5.1	Signal number to be used
Output	none	
Error	cc	Carry bit set
	d0.1	FASTBUS error code

ervice Request

er

ode (802)

ent number

e / 1 to enable

et

5 error code

ASTBUS Interrupt Message

er

ode (803)

ent number

e / 1 to enable

et

5 error code

storage information (used by `fbmon`)

er

ode (832)

info structure

et

5 error code

ASTBUS error name

mber

n code (833)

US error code

to error string

t set

US error code

processor instruction name

mber

n code (834)

essor instruction code

to instruction string

it set

US error code

/11/88

LEVEL */

R */