Aleph Event Builder
FASTBUS library

A. Castro, A. Miotto.

This paper has: 24 pages figures annexes.
If you wish to receive it, please send your request to:
A. Mazzari - EP
Aleph Event Builder
FASTBUS library

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. Miotti.
Network Address: AL0VOL::MIOTTO at CERN
Version of Document: 1.0
Revision date: 16 June 1988
Status: Draft

Changes from last version:

This is a new document.
Contents

INTRODUCTION 1

FASTBUS ROUTINES
1. ENVIRONMENT MANAGEMENT 4
2. OPERATIONAL PARAMETERS 6
3. SIMPLE TRANSACTION ROUTINES
   3.1 Single data word transfer 8
   3.2 Block transfers 9
   3.3 Secondary address routines 11
4. COMPOUND TRANSACTION ROUTINES
   4.1 Access Segment Interconnect Route Table 13
   4.2 Read-Modify-Write FASTBUS locations 13
5. SYNCHRONIZATION, SYSTEM RESOURCE AND PORT ROUTINE 15
6. FASTBUS SR AND INTERRUPT MESSAGE ROUTINES 16
7. STATUS AND ERROR HANDLING 19

ERROR CODES 21

APPENDIX
A. LIST OF RESERVED NAMES (sorted by short name) 22

TEST PROGRAM 24
INTRODUCTION

The main feature of the Aleph Event Builder is a 68020 coprocessor that allows FASTBUS actions to be executed as single machine instructions. The coprocessor supports up to 16 simultaneous users and most of the standard FASTBUS routines.

A driver and a library have been written to allow multi-user utilization of the coprocessor under the OS-9/68k operating system. The driver provides exception and interrupt handling, and is called by the library only for those routines that require some protection from user access (environment management and interrupt connection routines). All FASTBUS actions are performed without driver calls to avoid software overhead: the library mainly interfaces high level languages to assembler coprocessor instructions.

This means that even if the implementation follows as much as possible, the standard defined by U.S. NIM Committee (1), some differences are inevitable in order to optimize the use of the coprocessor and to keep software overhead to a minimum.

DISTRIBUTION The library contains entry points for FORTRAN and C languages. Calling conventions for C routines are the following: parameters are normally passed by value; the FASTBUS error code is the function return value, while additional values are returned using parameters passed by reference.

For the use of the library, the following files are distributed:

a) fastbus an OS9 device driver descriptor;
b) fb the FASTBUS exception handling driver;
c) fbmon FASTBUS exception monitor
d) fb.l library for FASTBUS hardware interfacing;
e) cfbdef.h C include file with FASTBUS definitions;
f) ffbdef.inc FORTRAN include file with FASTBUS definitions;
g) test.c source of an example program in C language;
h) test executable of test.c;
i) makefile to be used for compiling C programs;

INSTALLATION After the OS9 boot, the following commands should be executed (for example from a startup file):
$ load /{path_name}/fb       ! load driver in memory
$ load /{path_name}/fastbus  ! load descriptor in memory
$ iniz fastbus               ! initialize it

Coprocessor microcode version 2.23 or greater has to be used in order to run interrupt routines properly.

**PROGRAMMING RULES** The main program must contain a call to **fb_open** before any other call to FASTBUS routines. An environment with identifier **FB_DEFAULT_EID** is created and initialised.

The routine **fcienv** can be used to create up to a maximum of 16 FASTBUS environments.

After each FASTBUS action, the return value should always be examined by the user. In case of values different from **FENORM** the **ferrpt** routine will output available information on the standard error output path. Automatic report is not implemented.

At the end of each session, the routine **fb_close** should be called before exiting.

Only one Service Request connection is allowed. For this reason FORTRAN entry points for SR connections are not defined, and privileges are required to make the connection.

Only one FASTBUS Interrupt Message connection per task, and one connection per receiver block number is allowed (this means that two tasks can not connect to the same block).

**NAMING CONVENTIONS** Most FASTBUS names considered in this implementation are defined in both short and long form; error codes and a few other names are defined only in the short form, excepted the routine names **fopen** and **fclose**, for which an incompatibility would have arisen with the standard C library routines for opening and closing files; the long names **fb_open** and **fb_close** are used instead. Refer to the appendix for the list of reserved names.

**COMPILATION OF C PROGRAMS** The include file **cfbdef.h** should be placed in the **DEFS** subdirectory of the default device and the library **fb.lib** in the **LIB** subdirectory. If the **make** utility is loaded, a source program requiring only **fb.lib** and the standard C libraries can be compiled with the command:

$ make T=(file_name)               ! without the .c extension

**THE TEST PROGRAM** This is a very simple program showing the use of several FASTBUS calls in C. The listing is included at the end of this manual.

**ENVIRONMENT RECOVERY** An exit handler takes care of releasing environments if the user does not close the session or if the program is prematurely aborted. With OS-9 V2.1 some situations arose in which the exit handler was not properly called by the system; this seems to have been fixed with OS-9 V2.2. Anyway, to recover from these situations be sure that all process using FASTBUS are stopped and then type:

$ deiniz fastbus
$ iniz fastbus
DIFFERENCES FROM THE STANDARD  Any difference from the Standard FASTBUS software is marked with one ore more † symbols in the following. These conditions can be met:

NOT IMPLEMENTED means a category "A" (mandatory) routine that has not been implemented. The only routine that could not be implemented without avoiding inacceoptable overheads is $SGSUM$ (decode summary status). Other missing routines will be implemented in following releases.

EXTENSION means a routine or a parameter not defined in the standard and meaningful only in this implementation.

NON-STANDARD means that the specified routine or parameter has been modified from the standard definitions to optimize its use.

(1) U.S. NIM Committee - FASTBUS standard routines - March 1987 DOE/ER-0325
FASTBUS ROUTINES

1. ENVIRONMENT MANAGEMENT

/* C CALLS */
#include <cfbdef.h>
FB_environment_id id;
FB_error_code iret;
FB_word env [FPENVW];

C FORTRAN CALLS
INCLUDE 'FFBDEF.INC'
INTEGER*4 ID, IRET, ENV(FPENVW)

FB_OPEN Open a FASTBUS session.
Syntax:  iret = fb_open ();
          CALL FB_OPEN(IRET) !or CALL FOPEN(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.
Syntax:  iret = fb_close ();
          CALL FB_CLOSE(IRET) !or CALL FCLOSE(IRET)
Description: When use of FASTBUS is no longer required, the user shall call this routine.

FCIENV Create an immediate execution FASTBUS environment.
Syntax:  iret = fcienv (id);
          CALL FCIENV(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.
Syntax:  iret = frlnv (id);
          CALL FRLENV(IRET, ID)
Description: Release the environment with identifier id.
**FRSENV**  Reset a FASTBUS environment.
Syntax: \[ \text{iret} = \text{frsenv} (\text{id}); \]
\[ \text{CALL FRSENV(IRET, ID)} \]
Description: Reset the environment with identifier \text{id} to the default values.

**FSTENV**  Set a FASTBUS environment.
Syntax: \[ \text{iret} = \text{fstenv} (\text{id, env, FPENVS}); \]
\[ \text{CALL FSTENV(IRET, ID, ENV, FPENVS)} \]
Description: Set the environment with identifier \text{id}. \text{env} is the pointer to a 15 words array.

**FGTENV**  Get a FASTBUS environment.
Syntax: \[ \text{iret} = \text{fgtenv} (\text{id, env, FPENVS, FPENVS}); \]
\[ \text{CALL FSTENV(IRET, ID, ENV, FPENVS, FPENVS)} \]
Description: Returns in \text{env} the environment \text{id} parameters.
2. OPERATIONAL PARAMETERS

/* C CALLS */
#include <cfbdif.h>
FB_environment id;
FB_error_code iret;
FB_parameter_id par_id;
FB_parameter_value par_val;

C FORTRAN CALLS
#include 'FFBDEF.INC'
INTEGER*4 ID, IRET, PAR_ID, PAR_VAL

FBPINIT†
Initialize FASTBUS operational parameters.

FBPSET Set FASTBUS operational parameter.
Syntax: iret = fbpset (id, par_id, par_val);
CALL FBPSET(IRET,ID,PARM_ID,PARM_VAL)
Description: Assigns par_val to the operational parameter specified by par_id.

FBPGET Get FASTBUS operational parameter.
Syntax: iret = fbpget (id, par_id, &par_val);
CALL FBPGET(IRET,ID,PARM_ID,PARM_VAL)
Description: Reads into par_val the operational parameter specified by par_id.

The operational parameters implemented are:
- FPARBL Arbitration level - default value is assigned by the driver.
- FPEXTH Exit severity threshold - default value is FB_SEV_ERROR. This parameter is checked inside the fstart routine only, so the program will not abort after an error if fstart is not called.
- FPENV과 Size in bytes of the environment - fixed value is 60.
- FPENVtv† Size in longwords of the environment - fixed value is 15.
- FPNOWT Do not wait for completion of action (only valid for block transfer actions) - default value FB_TRUE††.
- FPPRIV†† FASTBUS privileges. This parameter can be set only if the process owner is OS-9 Super User. Valid privileges are:
  - BUSRST: may issue a FASTBUS reset signal
  - SRVCON: may connect to SR interrupts
- FPPRTY Control of parity generation - default value is FB_PARITY_NONE.
† NOT IMPLEMENTED
‡‡ EXTENSION
‡‡‡ NON-STANDARD: The standard default value is FB_FALSE.
3. SIMPLE TRANSACTION ROUTINES

3.1 Single data word transfer

C CALLS *
#include <cfbdef.h>
FB_environment id;
FB_error_code iret;
int prim_add, sec_add, sw_buf;

C FORTRAN CALLS
#include 'FFBDEF.INC'
INTEGER*4 ID, IRET, PRIM_ADD, SEC_ADD, SW_BUF

FRC Read single word from Control Space.
Syntax: iret = frc (id, prim_add, sec_add, FBVAR, &sw_buf);
CALL FRC(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)
Description: Moves a 32 bit word from the Primary Address prim_add, Secondary Address sec_add to sw_buf.

FWC Write single word to Control Space.
Syntax: iret = fwc (id, prim_add, sec_add, FBVAR, sw_buf);
CALL FWC(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)
Description: Moves the 32 bit word sw_buf to the Primary Address prim_add and Secondary Address sec_add.

FRD Read single word from Data Space.
Syntax: iret = frd (id, prim_add, sec_add, FBVAR, &sw_buf);
CALL FRD(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)
Description: Moves a 32 bit word from the Primary Address prim_add, Secondary Address sec_add to sw_buf.

FWD Write single word to Data Space.
Syntax: iret = fwd (id, prim_add, sec_add, FBVAR, sw_buf);
CALL FWD(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)
Description: 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.  
Syntax:  
\[
\text{iret = frcm (id, prim_add, sec_add, FBVAR, &sw_buf);} 
\]
\[
\text{CALL FRCM(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)} 
\]
Description:  
Moves a 32 bit word from the Primary Address \text{prim_add}, Secondary Address \text{sec_add} to \text{sw_buf}.

**FWCM**  
Write single word to Control Space Multi-listener.  
Syntax:  
\[
\text{iret = fwcm (id, prim_add, sec_add, FBVAL, sw_buf);} 
\]
\[
\text{CALL FWCM(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)} 
\]
Description:  
Moves the 32 bit word \text{sw_buf} to the Primary Address \text{prim_add} and Secondary Address \text{sec_add}.

**FRDM**  
Read single word from Data Space Multi-listener.  
Syntax:  
\[
\text{iret = frdm (id, prim_add, sec_add, FBVAR, &sw_buf);} 
\]
\[
\text{CALL FRDM(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)} 
\]
Description:  
Moves a 32 bit word from the Primary Address \text{prim_add}, Secondary Address \text{sec_add} to \text{sw_buf}.

**FWDM**  
Write single word to Data Space Multi-listener.  
Syntax:  
\[
\text{iret = fwdm (id, prim_add, sec_add, FBVAL, sw_buf);} 
\]
\[
\text{CALL FWDM(IRET, ID, PRIM_ADD, SEC_ADD, FBVAR, SW_BUF)} 
\]
Description:  
Moves the 32 bit word \text{sw_buf} to the Primary Address \text{prim_add} and Secondary Address \text{sec_add}.

**3.2 Block transfers**

```c
/* C CALLS */
#include <cfbdef.h>
FB_environment id;
FB_error_code iret;
int prim_add, sec_add, *buffer, byte_count;

C FORTRAN CALLS
INQUIRE 'FFBDEF.INC'
INTEGER*4 ID, IRET, PRIM_ADD, SEC_ADD, @BUFFER, BYTE_COUNT
```
**FRCB**
Block transfer read from Control Space.

Syntax:
```c
iret = frcb (id, prim_add, sec_add, FBVAR, buffer, byte_count);
call frcb (iret, id, prim_add, sec_add, fbvar, buffer, byte_count)
```

Description: Transfers `byte_count` bytes from the Primary Address `prim_add`, Secondary Address `sec_add`, to the module location `buffer`.

**FWCB**
Block transfer write to Control Space.

Syntax:
```c
iret = fwcb (id, prim_add, sec_add, FBVAR, buffer, byte_count);
call fwcb (iret, id, prim_add, sec_add, fbvar, buffer, byte_count)
```

Description: Transfers `byte_count` bytes from the location `buffer` to Primary Address `prim_add`, Secondary Address `sec_add`.

**FRDB**
Block transfer read from Data Space.

Syntax:
```c
iret = frdb (id, prim_add, sec_add, FBVAR, buffer, byte_count);
call frdb (iret, id, prim_add, sec_add, fbvar, buffer, byte_count)
```

Description: Transfers `byte_count` bytes from the Primary Address `prim_add`, Secondary Address `sec_add`, to the module location `buffer`.

**FWDB**
Block transfer write to Data Space.

Syntax:
```c
iret = fwdb (id, prim_add, sec_add, FBVAR, buffer, byte_count);
call fwdb (iret, id, prim_add, sec_add, fbvar, buffer, byte_count)
```

Description: Transfers `byte_count` bytes from the location `buffer` to Primary Address `prim_add`, Secondary Address `sec_add`.

**FRCBM**
Block transfer read from Control Space, Multi-listener.

Syntax:
```c
iret = frcbm (id, prim_add, sec_add, FBVAR, buffer,
               byte_count);
call frcbm (iret, id, prim_add, sec_add, fbvar, buffer, byte_count)
```

Description: Transfers `byte_count` bytes from the Primary Address `prim_add`, Secondary Address `sec_add`, to the module location `buffer`.

**FWCBM**
Block transfer write to Control Space, Multi-listener.

Syntax:
```c
iret = fwcbm (id, prim_add, sec_add, FBVAR, buffer,
              byte_count);
call fwcbm (iret, id, prim_add, sec_add, fbvar, buffer, byte_count)
```

Description: Transfers `byte_count` bytes from the location `buffer` to Primary Address `prim_add`, Secondary Address `sec_add`.
FRDBM  Block transfer read from Data Space, Multi-listener.

Syntax:  iret = frdbm (id, prim_add, sec_add, FBVAR, buffer,  
          byte_count);

CALL FRDBM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFER,BYTE_COUNT)

Description: Transfers byte_count bytes from the Primary Address prim_add, Secondary Address  
sec_add, to the module location buffer.

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

Syntax:  iret = fwdbm (id, prim_add, sec_add, FBVAR, buffer,  
          byte_count);

CALL FWDBM(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFER,BYTE_COUNT)

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

FIRDB† Indirect block transfer read from Data Space.

Syntax:  iret = firdb (id, prim_add, sec_add, FBVAR, buffer, max_count);

CALL FIRDB(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,BUFER,MAX_COUNT)

Description: A single word read from Primary Address prim_add, Secondary Address sec_add is  
performed: 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 performed: 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

/* C CALLS */
#include <cftbdf.h>
FB_environment id;
FB_error_code iret;
int prim_add, sw_buf;

C FORTRAN CALLS
 INCLUDE 'FFBDEF.INC'
 INTEGER*4 ID, IRET, PRIM_ADD, SW_BUF
**FRCSA**  
Read NTA register in Control Space.

Syntax:  
```c
iret = frcsa (id, prim_add, FBVAR, &sw_buf);
CALL FRCSA(IRET, ID, PRIM_ADD, FBVAR, SW_BUF)
```

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

---

**FWCSA**  
Write NTA register in Control Space.

Syntax:  
```c
iret = fwcsa (id, prim_add, FBVAL, sw_buf);
CALL FWCSA(IRET, ID, PRIM_ADD, FBVAR, SW_BUF)
```

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

---

**FRDSA**  
Read NTA register in Data Space.

Syntax:  
```c
iret = frdsa (id, prim_add, FBVAR, &sw_buf);
CALL FRDSA(IRET, ID, PRIM_ADD, FBVAR, SW_BUF)
```

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

---

**FWDSA**  
Write NTA register in Data Space.

Syntax:  
```c
iret = fwdsa (id, prim_add, FBVAL, sw_buf);
CALL FWDSA(IRET, ID, PRIM_ADD, FBVAR, SW_BUF)
```

Description:  
Writes the NTA register with the 32 bit word `sw_buf` at Primary Address `prim_add`. 
4. COMPOUND TRANSACTION Routines

4.1 Access Segment Interconnect Route Table

    /* C CALLS */
    #include <cfbdef.h>
    FB_environment id;
    FB_error_code iret;
    int prim_add, rt_add, sw_buf;

    C FORTRAN CALLS
    INCLUDE 'FFBDEF.INC'
    INTEGER*4 ID, IRET, PRIM_ADD, RT_ADD, SW_BUF

    FWRT Write SI Route Table.
    Syntax: iret = fwrt (id, prim_add, rt_add, FBVAL, sw_buf);
    CALL FWRT(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,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.
    Syntax: iret = frrt (id, prim_add, rt_add, FBVAR, &sw_buf);
    CALL FRRT(IRET,ID,PRIM_ADD,RT_ADD,FBVAR,SW_BUF)
    Description: Reads into sw_buf the entry indexed by rt_add in the SI at Primary Address prim_add.

4.2 Read-Modify-Write FASTBUS locations

    /* C CALLS */
    #include <cfbdef.h>
    FB_environment id;
    FB_error_code iret;
    int prim_add, sec_add, sec_add_0, sec_add_1, data_compare,
    data_compare_0, data_compare_1, data_update, data_update_0,
    data_update_1;

    C FORTRAN CALLS
    INCLUDE 'FFBDEF.INC'
    INTEGER*4 ID, IRET, PRIM_ADD, SEC_ADD, SEC_ADD_0, SEC_ADD_1,
    1 DATA_COMPARE, DATA_COMPARE_0, DATA_COMPARE_1, DATA_UPDATE,
    1 DATA_UPDATE_0, DATA_UPDATE_1;
FCASC† Compare and swap single word from Control Space.

Syntax: 
```asm
iret = fcasc (id, prim_add, sec_add, FBVAL, data_compare,
data_update);
```
```asm
CALL FCASC(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,DATA_COMPARE,
1       DATA_UPDATE)
```

Description: 
Compares the 32 bit word at Primary Address prim_add, Secondary Address sec_add with the word data_compare. If they are equal, substitutes the word with data_update. If they are not equal, stores the word in data_compare.

FCASD† Compare and swap single word from Data Space.

Syntax: 
```asm
iret = fcasd (id, prim_add, sec_add, FBVAL, data_compare,
data_update);
```
```asm
CALL FCASD(IRET,ID,PRIM_ADD,SEC_ADD,FBVAR,DATA_COMPARE,
1       DATA_UPDATE)
```

Description: 
Compares the 32 bit word at Primary Address prim_add, Secondary Address sec_add with the word data_compare. If they are equal, substitutes the word with data_update. If they are not equal, stores the word in data_compare.

FCASC2† Compare and swap two words from Control Space.

Syntax: 
```asm
iret = fcasc2 (id, prim_add, sec_add_0, sec_add_1, FBVAL,
data_compare_0, data_update_0, data_compare_1, data_update_1);
```
```asm
CALL FCASC2(IRET,ID,PRIM_ADD,SEC_ADD_0,SEC_ADD_1,FBVAR,
1       DATA_COMPARE_0, DATA_UPDATE_0, DATA_COMPARE_1, DATA_UPDATE)
```

Description: 
Compares the 32 bit words at Primary Address prim_add, Secondary Address sec_add_0 and sec_add_1 with the words data_compare_0 and data_compare_1 respectively. If both words are equal, substitutes them with data_update_0 and data_update_1 respectively. If a word is not equal, stores the words in data_compare_0 and data_compare_1.

FCASD2† Compare and swap two words from Data Space.

Syntax: 
```asm
iret = fcasd2 (id, prim_add, sec_add_0, sec_add_1, FBVAL,
data_compare_0, data_update_0, data_compare_1, data_update_1);
```
```asm
CALL FCASD2(IRET,ID,PRIM_ADD,SEC_ADD_0,SEC_ADD_1,FBVAR,
1       DATA_COMPARE_0, DATA_UPDATE_0, DATA_COMPARE_1, DATA_UPDATE)
```

Description: 
Compares the 32 bit words at Primary Address prim_add, Secondary Address sec_add_0 and sec_add_1 with the words data_compare_0 and data_compare_1 respectively. If both words are equal, substitutes them with data_update_0 and data_update_1 respectively. If a word is not equal, stores the words in data_compare_0 and data_compare_1.

† EXTENSION
5. SYNCHRONIZATION, SYSTEM RESOURCE AND PORT ROUTINE

/* C CALLS */
#include <cfbdef.h>
FB_environment id;
FB_error_code  iret;
int    slot;

C FORTRAN CALLS
  INCLUDE 'FFBDEF.INC'
  INTEGER*4 ID, IRET, SLOT

FCOMWT  Wait for completion of operation.
Syntax:   iret = fcomwt (id);
          CALL FCOMWT(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:   iret = fwa i (FB_AEB_PORT, &slot);
          CALL FWA I(IRET,FB_AEB_PORT,SLOT)
Description: Reads into slot the geographical location of the station, where the module is located.

FBPRST†† Issue Reset FASTBUS.
Syntax:   iret = fbprst (FB_AEB_PORT);
          CALL FBPRST(IRET,FB_AEB_PORT)
Description: Issue FASTBUS Reset Bus signal on the master port.
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

/* C CALLS */
#include <cfbdef.h>
FB_error_code iret;
FB_integer rec_blk, flt_word;
FB_word flt_mask, flt_val;
int (*procSR)(), (*procFIR)();
/* CONNECTED ROUTINES:
** procSR (SR_source)
** int SR_source;
**
** procFIR (&rec_blk, mess_buffer, &mess_length, &port) 1
** int rec_blk, *mess_buffer, mess_length, port;
*/

C FORTRAN CALLS
INCLUDE 'FFBDEF.INC'
INTEGER*4 IRET, REC_BLK, FLT_WORD, FLT_MASK, FLT_VAL
EXTERNAL PROCFIR
C CONNECTED ROUTINE
C SUBROUTINE PROCFIR(REC_BLK,MESS_BUFFER,MESS_LENGTH,PORT)
C INTEGER*4 REC_BLK, *MESS_BUFFER, MESS_LENGTH, PORT

FBSRC Connect routine to SR.
Syntax: iret = fbsrc (FB_SR_DEFAULT, FB_AEB_PORT, procSR);
FORTRAN CALL NOT AVAILABLE
Description: When an SR occurs the routine procSR is called if the port is enabled, and
FB_SR_DEFAULT is passed as parameter. It is the user responsibility to find and reset the
SR source(s). Only one user can connect to the SR interrupt.
Notes: SRVCON privilege is required.

FBSRD Disconnect routine from SR.
Syntax: iret = fbsrd (FB_SR_DEFAULT, FB_AEB_PORT);
FORTRAN CALL NOT AVAILABLE
Description: The connection established by fbsrc is broken.
Notes: SRVCON privilege is required.

1 The operator ADDRESS OF can not be used inside a function call, and the syntax should be
procFIR (rb_ptr, ...
int *rb_ptr, ...
In a next release procFIR parameters will be passed by value, and so the 's and this note will disappear.
**FBSREN**  
Enable SR connections.  
**Syntax:**  
\[ \text{iret} = \text{fbsren\ (FB\_AEB\_PORT);} \]  
FORTRAN CALL NOT AVAILABLE  
**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.  
**Syntax:**  
\[ \text{iret} = \text{fbsrds\ (FB\_AEB\_PORT);} \]  
FORTRAN CALL NOT AVAILABLE  
**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 FIR.  
**Syntax:**  
\[ \text{iret} = \text{fbfirc\ (FB\_ENV\_PORT, rec\_blk, flt\_mask, flt\_val,} \]  
\[ \text{flt\_word, procFIR);} \]  
CALL FBFIRC(IRET,FB_ENV_PORT,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.

**FBFIRD**  
Disconnect routine from FIR.  
**Syntax:**  
\[ \text{iret} = \text{fbfird\ (FB\_ENV\_PORT, rec\_blk, flt\_mask, flt\_val,} \]  
\[ \text{flt\_word, procFIR);} \]  
CALL FBFIRD(IRET,FB_ENV_PORT,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.

**FBFIRE**  
Enable FIR connections.  
**Syntax:**  
\[ \text{iret} = \text{fbfire\ (FB\_ENV\_PORT);} \]  
CALL FBFIRE(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.
Syntax:   iret = fbfirs (FB_ENV_PORT);
          CALL FBFIRS(IRET,FB_ENV_PORT)
Description:  The connected routine is not called in response to FASTBUS Interrupt Messages after this routine has been called.
7. STATUS AND ERROR HANDLING

/* C CALLS */
#include <cfbdef.h>
FB_environment id;
FB_error_code  iret0, iret;
FB_associated_parameter *ass_par;
FB_where_occurred  *wh_occ;

C FORTRAN CALLS
#include 'FFBDEF.INC'
INTEGER*4 ID, IRETO, IRET, @ASS_PAR, @WH_OCC

FSGSUM† Decode summary status.

FSFSUP†† Find supplementary status information.
Syntax:  
iret0 = fsfsup (id, iret, &ass_par, &wh_occ);
CALL FSFSUP(IRETO,ID,IRET,&ASS_PAR,&WH_OCC)

Description: 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:

FSRPT†† Report a FASTBUS error
Syntax:  
iret0 = fsrpt (id, iret, ass_par, wh_occ);
CALL FSRPT(IRETO,ID,IRET,ASS_PAR,WH_OCC)

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

The associated parameter and where_occurred structures are defined as follows

struct associated_parameter {  
    int  type;
    int id;
    int error_code;
    int severity_level;
    char *error_name;
    int cp_status; /*MEANINGFUL ONLY IF type>0 */
    char *instr_name; /*MEANINGFUL ONLY IF type>0 */
    int primary_address; /*MEANINGFUL ONLY IF type>1 */
    int secondary_address; /*MEANINGFUL ONLY IF type>1 */
    int address_register; /*MEANINGFUL ONLY IF type=3 */
    int byte_counter; } /*MEANINGFUL ONLY IF type=3 */
struct where_occurred {
    char *routine_name;
    int pc_at_exception; } /*MEANINGFUL ONLY IF type>0 */

† NOT IMPLEMENTED

†† NON-STANDARD: The standard types for the associated_parameter and where_occurred parameters are 32 bit integer values.
ERROR CODES

- The following standard error codes are defined:

<table>
<thead>
<tr>
<th>FEACON</th>
<th>FEAKTO</th>
<th>FEASS1</th>
<th>FEASS2</th>
<th>FEASS3</th>
<th>FEASS4</th>
<th>FEASS5</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEASS6</td>
<td>FEASS7</td>
<td>FEBUF</td>
<td>FEBSS2</td>
<td>FECLSD</td>
<td>FECON</td>
<td>FEDCON</td>
</tr>
<tr>
<td>FEDKTO</td>
<td>FEDPE</td>
<td>FEDSS1</td>
<td>FEDSS2</td>
<td>FEDSS3</td>
<td>FEDSS4</td>
<td>FEDSS5</td>
</tr>
<tr>
<td>FEDSS6</td>
<td>FEDSS7</td>
<td>FEEIOV</td>
<td>FEEREL</td>
<td>FEINEI</td>
<td>FEIPRV</td>
<td>FENCON</td>
</tr>
<tr>
<td>FENORM</td>
<td>FENPRV</td>
<td>FEOPEN</td>
<td>FESATO</td>
<td>FESSS1</td>
<td>FESSS2</td>
<td>FESSS3</td>
</tr>
<tr>
<td>FESSS4</td>
<td>FESSS5</td>
<td>FESSS6</td>
<td>FESSS7</td>
<td>FEUPAR</td>
<td>FEWTTO</td>
<td></td>
</tr>
</tbody>
</table>

- The following standard errors codes have a special meaning:

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

- In addition these new codes have been introduced:

**FB_ERR_ENV_NOT_INITIALIZED**

Short name: **FEENIN**,  Severity: **FSERR**

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.

**FB_ERR_PRIMARY_ADDRESS_PARITY_ERROR**

Short name: **FEAPE**,  Severity: **FSERR**

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

**FB_ERR_SECONDARY_ADDRESS_PARITY_ERROR**

Short name: **FEAPE**,  Severity: **FSERR**

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

**FB_ERR_ARBITRATION_TIMEOUT**

Short name: **FEGKTO**,  Severity: **FSERR**

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

LIST OF RESERVED NAMES (sorted by short name)

<table>
<thead>
<tr>
<th>short name</th>
<th>long name</th>
</tr>
</thead>
<tbody>
<tr>
<td>fb_close¹</td>
<td>FEAKTO</td>
</tr>
<tr>
<td>fb_open</td>
<td>FEAPE</td>
</tr>
<tr>
<td>FBAEBP</td>
<td>FB_AEB_PORT</td>
</tr>
<tr>
<td>FBDEID</td>
<td>FB_DEFAULT_EID</td>
</tr>
<tr>
<td>FBEENV</td>
<td>FB_ENV_PORT</td>
</tr>
<tr>
<td>FBFIRC</td>
<td>FB_FIR_CONNECT</td>
</tr>
<tr>
<td>fbfirc</td>
<td>fb_fir_connect</td>
</tr>
<tr>
<td>FBFIRD</td>
<td>FB_FIR_DISCONNECT</td>
</tr>
<tr>
<td>fbfird</td>
<td>fb_fir_disconnect</td>
</tr>
<tr>
<td>FBFIRE</td>
<td>FB_FIR_ENABLE</td>
</tr>
<tr>
<td>fbfire</td>
<td>fb_fir_enable</td>
</tr>
<tr>
<td>FBFIRS</td>
<td>FB_FIR_DISABLE</td>
</tr>
<tr>
<td>fbfirs</td>
<td>fb_fir_disable</td>
</tr>
<tr>
<td>FBINID</td>
<td>FB_INVALID_EID</td>
</tr>
<tr>
<td>FBPGET</td>
<td>FB_PAR_GET</td>
</tr>
<tr>
<td>fbpget</td>
<td>fb_par_get</td>
</tr>
<tr>
<td>FBPBSET</td>
<td>FB_PAR_RESET</td>
</tr>
<tr>
<td>fbpset</td>
<td>fb_par_set</td>
</tr>
<tr>
<td>fbsrc</td>
<td>fb_sr_connect</td>
</tr>
<tr>
<td>FBSDR</td>
<td>fb_sr_disconnect</td>
</tr>
<tr>
<td>FBSRDF</td>
<td>FB_SR_DEFAULT</td>
</tr>
<tr>
<td>fbsrds</td>
<td>fb_sr_disable</td>
</tr>
<tr>
<td>fbsren</td>
<td>fb_sr_enable</td>
</tr>
<tr>
<td>FBBVAL</td>
<td>FB_BUFFER_VAR</td>
</tr>
<tr>
<td>FBBVAR</td>
<td>FB_BUFFER_VAR</td>
</tr>
<tr>
<td>FCASC</td>
<td>FEGKTO</td>
</tr>
<tr>
<td>fcasc</td>
<td>FENEI</td>
</tr>
<tr>
<td>FCASC2</td>
<td>FEIPRV</td>
</tr>
<tr>
<td>fcas2c</td>
<td>FENCON</td>
</tr>
<tr>
<td>FCASD</td>
<td>FENORM</td>
</tr>
<tr>
<td>fasd</td>
<td>FENPRV</td>
</tr>
<tr>
<td>FCASD2</td>
<td>FEOOPS</td>
</tr>
<tr>
<td>fasd2</td>
<td>FOPEN</td>
</tr>
<tr>
<td>FCENV</td>
<td>FESAPE</td>
</tr>
<tr>
<td>FCREATE_IMMEDIATE_ENVIRONMENT</td>
<td>FESATO</td>
</tr>
<tr>
<td>fcienv</td>
<td>FESSS1</td>
</tr>
<tr>
<td>fb_create_immediate_environment</td>
<td>FESSS2</td>
</tr>
<tr>
<td>FCLOSE</td>
<td>FESSS3</td>
</tr>
<tr>
<td>fcomwt</td>
<td>FESSS4</td>
</tr>
<tr>
<td>fcomwt</td>
<td>FESSS5</td>
</tr>
<tr>
<td>FEAGEN</td>
<td>FESSS6</td>
</tr>
<tr>
<td>FEBU</td>
<td>FESSS7</td>
</tr>
<tr>
<td>FECON</td>
<td>FEPPAR</td>
</tr>
<tr>
<td>FECLSD</td>
<td>FEWTTO</td>
</tr>
<tr>
<td>FEKTO</td>
<td>FFALSE</td>
</tr>
<tr>
<td>FGETENV</td>
<td>FB_FALSE</td>
</tr>
</tbody>
</table>

¹ Lowercase names indicate C entry points, while the same name in uppercase are used for FORTRAN entry points.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>fgtenv</td>
<td>fb_get_environment</td>
<td>FWCSC</td>
</tr>
<tr>
<td>FIRDB</td>
<td></td>
<td>fwcsa</td>
</tr>
<tr>
<td>firdb</td>
<td></td>
<td>FWD</td>
</tr>
<tr>
<td>FOPEN</td>
<td>FB_OPEN</td>
<td>fwpw</td>
</tr>
<tr>
<td>FPARBL</td>
<td></td>
<td>FWDB</td>
</tr>
<tr>
<td>FPENV</td>
<td></td>
<td>fpwdb</td>
</tr>
<tr>
<td>FPENVW</td>
<td></td>
<td>FWDBM</td>
</tr>
<tr>
<td>FPEXTH</td>
<td></td>
<td>fpwdbm</td>
</tr>
<tr>
<td>FPNOVT</td>
<td></td>
<td>FWDM</td>
</tr>
<tr>
<td>FPPENVN</td>
<td>FB_PARITY_EVEN</td>
<td>fwpm</td>
</tr>
<tr>
<td>FPPNON</td>
<td>FB_PARITY_NONE</td>
<td>fwpf</td>
</tr>
<tr>
<td>FPPODD</td>
<td>FB_PARITY_ODD</td>
<td>fwps</td>
</tr>
<tr>
<td>FPPRIV</td>
<td></td>
<td>FWRT</td>
</tr>
<tr>
<td>FPPRTY</td>
<td></td>
<td>fwt</td>
</tr>
<tr>
<td>FRC</td>
<td>FB_READ_CSR</td>
<td>frc</td>
</tr>
<tr>
<td>FRCB</td>
<td>FB_READ_CSR_BLOCK</td>
<td>frcb</td>
</tr>
<tr>
<td>FRCBM</td>
<td>FB_READ_CSR_BLOCK.MULT</td>
<td>frcbm</td>
</tr>
<tr>
<td>FRCD</td>
<td></td>
<td>frcm</td>
</tr>
<tr>
<td>FRCSA</td>
<td>FB_READ_CSR_SA</td>
<td>frcsa</td>
</tr>
<tr>
<td>FRD</td>
<td>FB_READ_DAT</td>
<td>frd</td>
</tr>
<tr>
<td>FRDB</td>
<td>FB_READ_DAT_BLOCK</td>
<td>frdb</td>
</tr>
<tr>
<td>FRDBM</td>
<td>FB_READ_DAT_BLOCK.MULT</td>
<td>frdbm</td>
</tr>
<tr>
<td>FRDM</td>
<td>FB_READ_DAT.MULT</td>
<td>frdm</td>
</tr>
<tr>
<td>FRDSA</td>
<td>FB_READ_DAT_SA</td>
<td>frdsa</td>
</tr>
<tr>
<td>FRLENV</td>
<td>FB_RELEASE_ENVIRONMENT</td>
<td>frlenv</td>
</tr>
<tr>
<td>FRLRT</td>
<td>FB_READ_ROUTE_TABLE</td>
<td>frrt</td>
</tr>
<tr>
<td>FRRT</td>
<td></td>
<td>frsenv</td>
</tr>
<tr>
<td>FSERR</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FSFSUP</td>
<td>FB_FIND_SUPPLEMENTARY</td>
<td>fsf</td>
</tr>
<tr>
<td>FSFTL</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FSINFO</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FSRTPT</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FSUDD</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FSUCC</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FSTDENV</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FSWARN</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>TRUE</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FWAI</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FWC</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FWC</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FWCB</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FWCBM</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FWCM</td>
<td></td>
<td>fsf</td>
</tr>
<tr>
<td>FWCM</td>
<td></td>
<td>fsf</td>
</tr>
</tbody>
</table>

23
```c
#include <stdio.h>
#include <cfbdef.h>

FB_environment_id id;
FB_error_code iret;
FB_word env [FPENVW];
FB_associated *a_p;
FB_whereOccurred *w_o;

/* SIMPLE ROUTINE TO CHECK FASTBUS RETURN CODE */
fb_check (code)
int code;
{
    if (code != FENORM)
    {
        fsfsup (FBDEID, code, &a_p, &w_o); /* GET INFO */
        fsrpt (FBDEID, code, a_p, w_o);    /* REPORT ERROR */
        return (0);
    }
    else
        return (1);
}

main()
{
    int prim, sec;
    int csr0, slot, i;

    iret = fb_open (); /* OPEN SESSION */
    fb_check (iret);

    iret = fbpset (FBDEID, FPARBL, 6);  /* SET ARBITRATION LEVEL */
    fb_check (iret);

    iret = fgtenv (FBDEID, env, FPENVS, FPENVS);
    if (fb_check (iret))
        printf (" environment status word = \$%x \n", *env);

    iret = fwait (FB_AEB_PORT, &slot); /* FIND SLOT */
    if (fb_check (iret))
        printf (" EB is on slot \$%d \n", slot);

    printf (" Read Control Space operation:\n");
    printf (" primary address ?\n");
    scanf ("%d", &prim);
    printf (" secondary address ?\n");
    scanf ("%d", &sec);

    iret = frs (FBDEID, prim, sec, FBVAR, &csr0); /* READ CSR */
    if (fb_check (iret))
        printf ("CSR0 = \%x\n", csr0);

    iret = fb_close (); /* CLOSE SESSION */
}
```