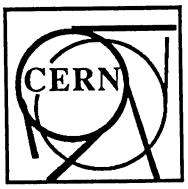
ALEPH 88-68
DATACQ 88-12
A. Miotto, A. Castro
17.6.1988



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

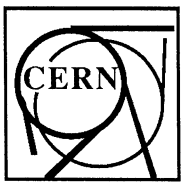
Aleph Event Builder
FASTBUS library

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

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

ALEPH 88-68
DATACQ 88-12
A. Miotto, A. Castro
17.6.1988



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

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: 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 languges. 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 `fsrpt` 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 arosen 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.l` in the `LIB` subdirectory. If the `make` utility is loaded, a source program requiring only `fb.l` 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 inacceptable overheads is `fsgsum` (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 = frlenv (id);`

`CALL FRLENV(IRET,ID)`

Description: Release the environment with identifier `id`.

FRSENV Reset a FASTBUS environment.

Syntax:
```
iret = frsenv (id);
CALL FRSENV(IRET,ID)
```

Description: Reset the environment with identifier `id` to the defult values.

FSTENV Set a FASTBUS environment.

Syntax:
```
iret = fstenv (id, env, FPENVS);
CALL FSTENV(IRET,ID,ENV,FPENVS)
```

Description: Set the environment with identifier `id`. `env` is the pointer to a 15 words array.

FGTENV Get a FASTBUS environment.

Syntax:
```
iret = fgtenv (id, env, FPENVS, FPENVS);
CALL FSTENV(IRET,ID,ENV,FPENVS,FPENVS)
```

Description: Returns in `env` the environment `id` parameters.

2. OPERATIONAL PARAMETERS

```
/* C CALLS */
#include <cfbdef.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
```

FBPINI† Initialize FASTBUS operational parameters.

FBPSET Set FASTBUS operational parameter.

Syntax: `iret = fbpset (id, par_id, par_val);`

`CALL FBPSET(IRET,ID,PAR_ID,PAR_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,PAR_ID,PAR_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 `fsrpt` routine only, so the program will not abort after an error if `fsrpt` is not called.
- `FPENVS` Size in bytes of the environment - fixed value is `60`.
- `FPENVW`†† 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, FBVAL, 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, FBVAL, 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:
```
iret = frcm (id, prim_add, sec_add, FBVAR, &sw_buf);
CALL FRCM(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`.

FWCM Write single word to Control Space Multi-listener.

Syntax:
```
iret = fwcm (id, prim_add, sec_add, FBVAL, sw_buf);
CALL FWCM(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`.

FRDM Read single word from Data Space Multi-listener.

Syntax:
```
iret = frdm (id, prim_add, sec_add, FBVAR, &sw_buf);
CALL FRDM(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`.

FWDM Write single word to Data Space Multi-listener.

Syntax:
```
iret = fwdm (id, prim_add, sec_add, FBVAL, sw_buf);
CALL FWDM(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`.

3.2 Block transfers

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

C FORTRAN CALLS
      INCLUDE 'FFBDEF.INC'
      INTEGER*4 ID, IRET, PRIM_ADD, SEC_ADD, @BUFFER, BYTE_COUNT
```

FRCB Block transfer read from Control Space.

Syntax:
```
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:
```
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:
```
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:
```
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:
```
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:
```
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,BUFFER,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,BUFFER,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,BUFFER,MAX_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

```
/* C CALLS */
#include <cfbdef.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:
```
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:
```
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:
```
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:
```
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:
```
iret = fcasc (id, prim_add, sec_add, FBVAL, data_compare,
              data_update);

 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:
```
iret = fcasd (id, prim_add, sec_add, FBVAL, data_compare,
              data_update);

 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:
```
iret = fcasc2 (id, prim_add, sec_add_0, sec_add_1, FBVAL,
 data_compare_0, data_update_0, data_compare_1, data_update_1);

 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`. 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:
```
iret = fcasd2 (id, prim_add, sec_add_0, sec_add_1, FBVAL,
 data_compare_0, data_update_0, data_compare_1, data_update_1);

 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`. 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 = fwai (FB_AEB_PORT, &slot);`

`CALL FWAI(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_lenght, &port)[1]
** int rec_blk, *mess_buffer, mess_lenght, 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_LENGHT,PORT)
C INTEGER*4 REC_BLK, @MESS_BUFFER, MESS_LENGHT, 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 responsability to find and reset the SR sorurce(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:
```
iret = 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:
```
iret = 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:
```
iret = fbfirc (FB_ENV_PORT, rec_blk, flt_mask, flt_val,
               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:
```
iret = fbfird (FB_ENV_PORT, rec_blk, flt_mask, flt_val,
               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:
```
iret = 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, IRET0, 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(IRET0,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(IRET0,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:

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

- The following standard errors codes have a special meaning:

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:

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: **FESAPE**, 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)

short name	long name
	fb_close[1]
	fb_open
FBAEBP	FB_AEB_PORT
FBDEID	FB_DEFAULT_EID
FBENVP	FB_ENV_PORT
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
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	
fcasc	
FCASC2	
fcasc2	
FCASD	
fcasd	
FCASD2	
fcasd2	
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	
FEFTL	
FEGKTO	
FEINEI	
FEIPRV	
FENCON	
FENORM	
FENPRV	
FEOOPS	
FEOPEN	
FESAPE	
FESATO	
FESSS1	
FESSS2	
FESSS3	
FESSS4	
FESSS5	
FESSS6	
FESSS7	
FEUPAR	
FEWTTO	
FFALSE	FB_FALSE
FGTENV	FB_GET_ENVIRONMENT

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

fgtenv	fb_get_environment
FIRDB	
firdb	
FOPEN	FB_OPEN
FPARBL	
FPENVS	
FPENVW	
FPEXTH	
FPNOWT	
FPPEVN	FB_PARITY_EVEN
FPPNON	FB_PARITY_NONE
FPPODD	FB_PARITY_ODD
FPPRIV	
FPPRTY	
FRC	FB_READ_CSR
frc	fb_read_csr
FRCB	FB_READ_CSR_BLOCK
frcb	fb_read_csr_block
FRCBM	FB_READ_CSR_BLOCK_MULT
frcbm	fb_read_csr_block_mult
FRCM	FB_READ_CSR_MULT
frcm	fb_read_csr_mult
FRCSA	FB_READ_CSR_SA
frcsa	fb_read_csr_sa
FRD	FB_READ_DAT
frd	fb_read_dat
FRDB	FB_READ_DAT_BLOCK
frdb	fb_read_dat_block
FRDBM	FB_READ_DAT_BLOCK_MULT
frdbm	fb_read_dat_block_mult
FRDM	FB_READ_DAT_MULT
frdm	fb_read_dat_mult
FRDSA	FB_READ_DAT_SA
frdsa	fb_read_dat_sa
FRLENV	FB_RELEASE_ENVIRONMENT
frlenv	fb_release_environment
FRRT	FB_READ_ROUTE_TABLE
frrt	fb_read_route_table
FRSENV	FB_RESET_ENVIRONMENT
frsenv	fb_reset_environment
FSERR	FB_SEV_ERROR
FSFSUP	FB_FIND_SUPPLEMENTARY
fsfsup	fb_find_supplementary
FSFTL	FB_SEV_FATAL
FSINFO	FB_SEV_INFO
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	
fwai	
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

test.c

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

FB_environment_id id;
FB_error_code iret;
FB_word env [FPENVW];
FB_associated_parameter *a_p;
FB_where_occurred *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 (" environmetn status word = $%x \n", *env);

     iret = fwai (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 ?");
     scanf ("%d",&prim);
     printf (" secondary address ?");
     scanf ("%d",&sec);

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

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