CSR

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CENTER FOR SPACE RESEARCH
CAMBRIDGE, MASSACHUSETTS 02139

REVISION LOG

TITLE: Software Detailed Design
FEP 10 Library

DOC. NO. 36-53223

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
—	1/16/95	—	all	New document	
A	5/1/95	36-253	all	Incorporate Review Comments	RFG 5/22/95
B	10/4/95	36-374	all	Remove FEP->BEP Request mailbox Changed append Data() to appendBlock() to support fix length block in ring buffer. R Removed FEP->BEP interrupt	5/27/97

34.0 FEP IO Library (36-53223 B)

34.1 Purpose

The FEP IO Library is a set of functions, executing in the FEP, which provide an interface to the front end hardware. Clients may use these functions to direct the FEP without having to know low level hardware details.

34.2 Uses

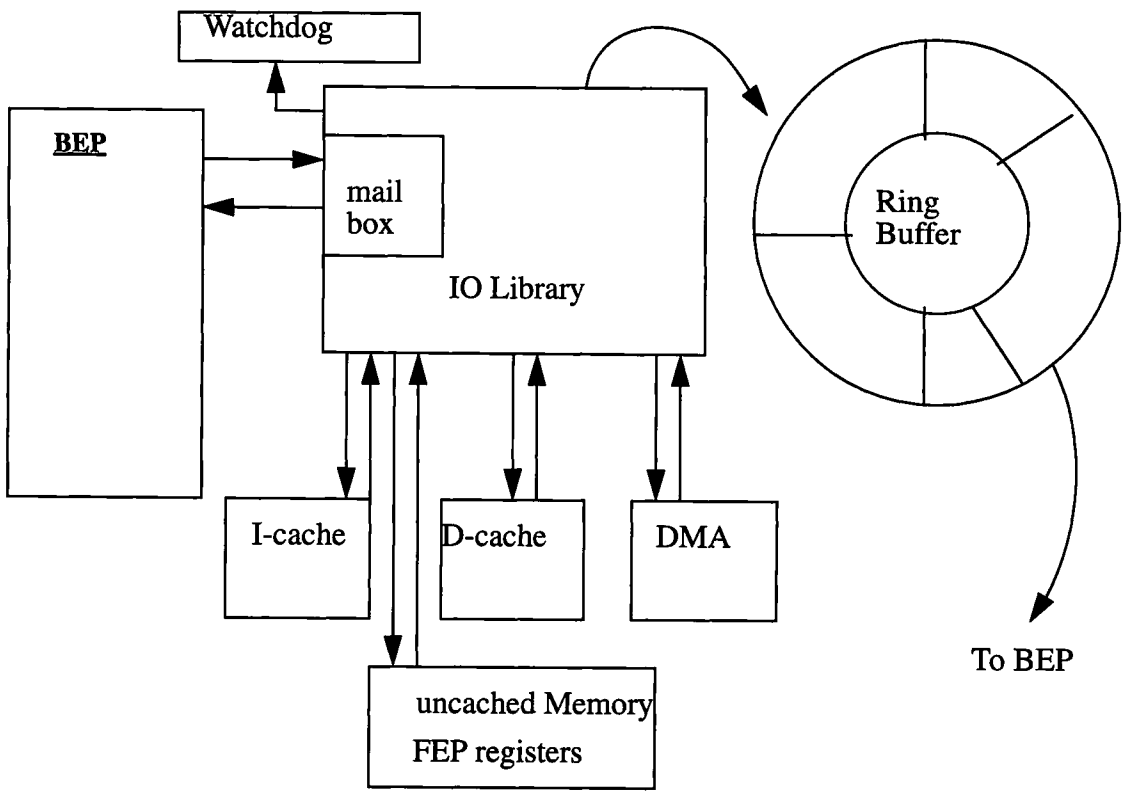
The FEP IO Library provides the following features:

Use 1:: Transfer data from the FEP to the BEP
Use 2:: Handle communications between the FEP and BEP
Use 3:: Manage the Mongoose DMA
Use 4:: Provide read/write access to internal Mongoose memory and status words
Use 5:: Reset the Watchdog Timer

34.3 Organization

Figure 152 illustrates the interfaces to the FEP provided by the IO Library.

FIGURE 152. FEP IO Library Interface Diagram



The FEP IO Library provides access functions to various areas of the FEP and BEP hardware.

- BEP/FEP - There are several aspects to the interface between the BEP and the FEP. The ring buffer is in shared memory. The FEP deposits data (usually image data or photon events) into the buffer in blocks less than or equal to 32-long words, which are in turn read out by the BEP (NOTE: If a given piece of data is less than 32-words in length, it is padded out to the full 32-word block size in the ring buffer).

  Another aspect of the interface is the communication protocol between the BEP and FEP. The method used is by a rudimentary mailbox. The BEP writes commands to the mailbox, and looks for replies in the same box.

  See the **ACIS Software Requirements Specification (#36-41001-02)** and the **ACIS Science Instrument Software Preliminary Design Specification (#36-02402)** for further details on the BEP/FEP hardware and software interface.

- ***FEP Internal*** - The other main function of the IO Library is to provide access to or control of the Mongoose internal memory. It is necessary to access the D-cache and I-cache, allow a method for executing a function in memory, and also to control the DMA for data transfers. There are also a set of basic access routines to some of the FEP hardware registers. A detailed description of the hardware registers and FEP memory can be found in the **DPA Hardware Specification & System Description #36-02104.**

- ***Watchdog*** - IO library also provides a function for maintaining the FEP watchdog timer. The timer must be reset periodically to avoid a hardware reset.

  See the **ACIS Software Requirements Specification (#36-41001-02)** and the **ACIS Science Instrument Software Preliminary Design Specification (#36-02402)** for further details on the watchdog timer.

34.4 Scenarios

The following paragraphs describe the basic function performed by the FEP, and which library functions should be used in a given scenario. In general, the functions prefaced by FIO (upper case) should be adequate for any task required by the science processing. All function prefaced with a lower case (fio) are used at the directive of the BEP.

34.4.1 Use 1: Transfer data from the FEP to the BEP

Transferring data across the interface uses a portion of the shared memory which has been configured as a ring buffer, consisting of a series of 32-long word blocks. The function **`FIOappendBlock()`** writes the specified data block to the ring buffer. This function does not return until the operation is completed, or until a command arrives at the BEP to FEP Command Mailbox. It is possible for the buffer to be full, so a wait may be required until the BEP has emptied more of the buffer. If the BEP is too slow in clearing the buffer, science processing may be affected. The function **`fioRbStatus()`** returns the status of the ring buffer and the available number of blocks. The library users do not need to call this routine before calling **`FIOappendBlock()`**.

34.4.2 Use 2: Handle communications between the FEP and BEP

The FEP has little independent action; therefore, there is a command protocol for directing the FEP. A portion of the shared memory is set aside for a mailbox. The function **`FIOgetNextCmd()`** polls the command mailbox to look for pending commands and performs the required actions. The functions **`fioPollMBox()`**, **`fioGetCmd()`**, and **`FIOwriteCmdReply()`** are used by **`FIOgetNextCmd()`** to handle the command interface. If the command is directed to the science processing, it returns that information to the calling function.

34.4.3 Use 3: Manage the FEP DMA

The Mongoose DMA may be used to transfer data between the D-cache and general purpose memory, or between two general memory locations. The function **`FIOdmaTransfer()`** initiates the transfer. The function **`FIOdmaDone()`** indicates when the transfer has completed.

34.4.4 Use 4: Provide read/write/execute access to internal FEP memory

The I-cache requires special access through read/write registers. The functions **`fioReadIcache()`** and **`fioWriteIcache()`** should be used to for any access to the I-cache. For completeness, the functions **`fioReadMem()`** and **`fioWriteMem()`** are available for accessing the D-cache. There are also several functions available to read and/or write some of the FEP hardware registers and access the addresses of the various por-

tions of hybrid memory, this includes the capability for directly excuting a function located in the memory.

34.4.5 Use 5: Reset WatchdogTimer

The watchdog timer must be reset frequently. If the timer reaches zero, it assumes the data processing is hung and resets the FEP. The function **`FIOtouchWatchdog()`** should be called within a **TBD** interval to reset the timer.

See the ACIS Software Preliminary Design Specification (FEP IO Library), (#36-02402) for further details on this interface.

34.5 BEP - FEP Communication Protocol

Communication between the BEP and the FEP is accomplished via two mailboxes, one for BEP initiated commands and their corresponding responses, the other for FEP initiated requests and their corresponding replies. The mailboxes are located in a section of shared memory between the BEP and the FEP. It is necessary to have some handshaking to ensure proper delivery and receipt of messages.

34.5.1 Mailbox Structure

The first field contains the mailbox state, the second contains the message. The actual message is divided into further sub-fields: the length of the following message, the type field, which indicates the kind of message, and any additional arguments required. Replies are written into the same structure as the initial command or request.

Command Mailbox

```
struct {
  MBOXstate state;
  struct {
     unsigned len;
     int type;
     unsigned args[SIZE];
  } COMMAND;
} CMD_MBOX;
*mailbox size is still 128 words
```

34.5.2 BEP to FEP

Communication originating in the BEP is usually a command directive to the FEP. There are two general types of commands, diagnostic and science. Diagnostic commands require information regarding the state of the FEP and are handled by the IO library. All science commands are forwarded to the science process.

The transfer sequence is as follows: the BEP writes a command to the mailbox and changes the state to NEW_MESSAGE. The FEP has been polling the mailbox (**FIOgetNextCmd()**) and takes the appropriate action when a command is received. In all cases, a response must be written to the command by the recipient (**FIOwriteCmdReply()**). This function will write the response and set the mailbox state to REPLY_RDY. The BEP reads the response and sets the mailbox state to NO_MESSAGE in preparation for the next exchange.

34.5.2.1 Diagnostic Command Formats

The following is a list of the commands that the FEP responds to. All other commands are forwarded to the science process.

34.5.2.1.1 Read I-Cache

This command accepts a source I-cache address and count of words to be read, in the format specified by Table 23 . If the parameters are within valid ranges, the data are read from I-Cache and stored back in the mailbox in the format specified by Table 24 . If the parameters are invalid, the action taken by the FEP is **TBD**. I

TABLE 23. Read I-Cache command format

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	2	Number of command arguments + type
type	Command type	enum	CMD_READ_ICACHE	Execute Read I-Cache command
args[0]	Address	32 bit unsigned	0x80080000 - 0x800ffffe	Source address in I-cache to read data from.
args[1]	Length	32 bit unsigned	0 -127	Number of words to be read. May not cross I-cache boundary

TABLE 24. Read I-Cache command response

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	0 - 127	Number of words in arguments field + type
type	Command type	enum	CMD_READ_ICACHE	Indicates which command has been executed
args[0 .. N]	Data	32 bit unsigned		Data values read from I-cache.

34.5.2.1.2 Write I-cache

This command accepts a destination address, count of words to be written, and data, in the format specified by Table 25 . If the parameters are within valid ranges, the data are writ-

ten into I-Cache. The command response is detailed in Table 26 . If the parameters are invalid, the action taken by the FEP is **TBD**.

TABLE 25. Write I-Cache command format

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	0 - 127	Number of command arguments + type
type	Command type	enum	CMD_WRITE_ICACHE	Execute Write I-Cache command
args[0]	Address	32 bit unsigned	0x80080000 - 0x800fffffe	Destination address in I-cache to write data too.
args[1.. N]	Data	32 bit unsigned		Data values to write to I-cache.

TABLE 26. Write I-Cache command response

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	1	Number of command arguments + type
type	Command type	enum	CMD_WRITE_ICACHE	Indicates which command has been executed

34.5.2.1.3 Read Memory

This command accepts a source address and count of words to be read, in the format specified by Table 27 . If the parameters are within valid ranges, the data are read from D-cache or general memory and stored back in the mailbox in the format specified by Table 28 . If the parameters are invalid, the action taken by the FEP is **TBD**.

TABLE 27. Read memory command format

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	2	Number of command arguments + type
type	Command type	enum	CMD_READ_MEM	Execute read memory command
args[0]	Address	32 bit unsigned	0x80000000 - 0x8007fffc 0xa0000000 - 0xfffffffc	Source address in memory to read data from.
args[1]	Length	32 bit unsigned	0 - 127	Number of words to be read. May not memory boundaries

TABLE 28. Read memory command response

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	0 - 127	Number of words in arguments field + type

TABLE 28. Read memory command response

Argument	Field	Units	Values	Description
type	Command type	enum	CMD_READ_MEM	Indicates which command has been executed
args[0 .. N]	Data	32 bit unsigned		Data values read from memory

34.5.2.1.4 Write Memory

This command accepts a destination address, count of words to be written, and data, in the format specified by Table 29 . If the parameters are within valid ranges, the data are written into memory. The command response is detailed in Table 30 . If the parameters are not valid, the action taken by the FEP is **TBD**.

TABLE 29. Write memory command format

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	0 - 127	Number of command arguments + type
type	Command type	enum	CMD_WRITE_MEM	Execute write memory command
args[0]	Address	32 bit unsigned	0x80000000 - 0x8007fffc 0xa0000000 - 0xfffffffc	Destination address in write data to.
args[1.. N]	Data	32 bit unsigned		Data values to write to memory.

TABLE 30. Write memory command response

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	1	Number of command arguments + type
type	Command type	enum	CMD_WRITE_MEM	Indicates which command has been executed.

34.5.2.1.5 Execute Memory

This command accepts the address of a function and a list of the functions arguments, in the format specified by Table 31 . The function is called with the indicated arguments. the return value of the function is stored back in the mailbox in the format specified by Table 32 .

TABLE 31. Execute memory command format

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	0 - 20	Number of command arguments + type
type	Command type	enum	CMD_EXECUTE_MEM	Execute "execute memory" command

TABLE 31. Execute memory command format

Argument	Field	Units	Values	Description
args[0]	Address	32 bit unsigned		Pointer to function to be called.
args[1..20]	Arguments	32 bit unsigned		Arguments for function call.

TABLE 32. Execute memory command response

Argument	Field	Units	Values	Description
len	Argument count	32 bit unsigned	2	Number of words in arguments field + type
type	Command type	enum	CMD_EXECUTE_MEM	Indicates which command has been executed.
args[0]	Data	32 bit unsigned		Return value from function call.

34.6 Specification

The following are the function definitions for the FEP IO Library. There are several access functions that will just return an address pointer, a hardware register value, or a constant. Most of these will be implemented as macros or in-line functions, defined in the file fep.h. See the **DPA Hardware Specification & System Description #36-02104, section TBD,** for more detailed information.

The functions defined in Section 34.6.1 through Section 34.6.19 are interface functions to read and write the FEP hardware registers. These are defined as in-line functions in the include file fep.h.

34.6.1 FIOgetBiasMapPtr()

Scope: Science

Return type **`unsigned *`**

Return a pointer to the start of the bias map. The address points to a pre-defined array of 1024 x 1024 16-bit pixels, and begins on a 4-byte boundary in non-cache memory.

34.6.2 FIOgetBiasConfig()

Scope: Science

Arguments:

unsigned *`*biasrec`*

unsigned `cnt`

Return type **`void`**

Retrieve the Bias Configuration information stored in I-Cache. `cnt` words are read from the I-cache address specified by the constant BIAS_CONFIG_SAVE and stored in the buffer indicated by `biasrec`.

34.6.3 FIOsetBiasConfig()

Scope: Science

Arguments

unsigned *`*biasrec`*

unsigned `cnt`

Return type **void**

Write *cnt* words of bias configuration data specified by *biasrec* into I-cache. The location in I-cache is specified by the constant BIAS_CONFIG_SAVE.

34.6.4 FIOgetBiasParityPlanePtr()

Scope: Science

Return type **unsigned ***

Return a pointer to the location of the bias parity plane previously specified by a call to **fioSetSegAllocReg()**.

34.6.5 FIOgetCcdRowStart()

Scope: Science

Return type **unsigned_**

Return the row index of the CCD where processing should begin, not necessarily the first row. The first row has index 0.

34.6.6 FIOsetCcdRowStart()

Scope: Science

Arguments

unsigned *row_index*

Return type **void_**

Set the index row of the CCD where processing should begin. The first row has index 0.

34.6.7 FIOgetImageMapPtr()

Scope: Science

Return type **unsigned ***

Return a pointer to the start of the Image map. The address points to a pre-defined array of 1024 x 1024 16-bit pixels, and begins on a 4-byte boundary in non-cache memory.

34.6.8 FIOgetImageMapRowIndex()

Scope: Science

Return type **unsigned**

Return an index to the last row written in the image map. The first map has index 0.

34.6.9 FIOgetImageMapRowLength()

Scope: Science

Return type **unsigned**

Return the count of CCD rows.

34.6.10 FIOsetImageMapRowLength()

Scope: Science

Arguments

unsigned *count*

Return type **void**

Set the number of CCD rows.

34.6.11 FIOgetImageMapRowStart()

Scope: Science

Return type **unsigned**

Return the Image Map row index at which the hardware should begin loading the image into non-cache memory. The first row has index 0.

34.6.12 FIOsetImageMapRowStart()

Scope: Science

Arguments

unsigned *row_index*

Return type **void**

Set the index row of the Image Map where we should begin loading the image. The first row has index 0.

34.6.13 FIOgetStartCntReg()

Scope: Science

Arguments

unsigned *regnum*

Return type **unsigned**

Return the contents of the indicated register (0 - 3).

34.6.14 FIOsetStartCntReg()

Scope: Science

Arguments

unsigned *regnum*

short *value*

Return type **void**

Set the indicated offset register (0 - 3) to the value specified.

34.6.15 FIOgetOverclockBufPtr()

Scope: Science

Return type **unsigned ***

Return a pointer to the start of the overclock buffer previously specified by a call to **fioSetSegAllocReg()**.

34.6.16 FIOgetThresholdRegister()

Scope: Science

Arguments

unsigned *regnum*

Return type **short**

Return the contents of the indicated threshold register (0 - 3).

34.6.17 FIOsetThresholdRegister()

Scope: Science

Arguments

unsigned *regnum*

short *thresholdval*

Return type **void**

Set the indicated threshold register (0 - 3) to the value specified. In order to support updating the threshold register values synchronously with the advent of the next exposure, this function stores the passed thresholdval into a holding variable. When the Beginning of Frame interrupt occurs, the interrupt handling routine will copy the contents of the holding variable into the actual threshold register.

NOTE: This scheme requires that the interrupt handler is capable of copying the holding variables into the threshold registers within one pixel clock of the Beginning of Frame interrupt (~10μseconds).

34.6.18 FIOgetThresholdXings()

Scope: Science

Return type **unsigned**

Return the value of the T-Plane crossings counter.

34.6.19 FIOgetTPlanePtr()

Scope: Science

Return type **`unsigned *`**

Return a pointer to the start of the T-Plane, previously specified by a call to **`fioSetSeg-AllocReg()`**.

34.6.20 FIOappendBlock()

#include fio.h

Scope: Science

Return type: **bool**

Arguments

unsigned **ptr*

unsigned *wordcnt*

Description:

Copy *wordcnt* words (32 bit values) pointed to by *ptr* to the block at the end of the ring buffer. *wordcnt* must be less than or equal to 32. This function does not return until the action is completed, or until a command is received at by the Command Mailbox. There may be delays if the ring buffer is full. Data values must be aligned on long word boundaries. While waiting for room the function maintains the watchdog timer using **FIOtouchWatchdog()**, and polls the status of the Command Mailbox using **fioPollMBox()**. If the block is sucessfully appended to the ring buffer, the function returns TRUE. If a command arrives while waiting for room in the ring buffer, the function aborts, and returns FALSE. Using **FIOgetNextCmd()**,the caller must then read and process the received command.

NOTE: Interrupts to the BEP on writes to an empty ring-buffer have been replaced by having the BEP poll the ring-buffers for data.

34.6.21 FIOdmaDone()

#include fio.h

Scope: Science

Return type **bool**

Arguments: none

Description:

This function returns TRUE if the most recent DMA transfer has completed. It returns FALSE if the transfer is not complete.

34.6.22 FIOdmaTransfer()

#include fio.h

Scope: Science

Return type **void**

Arguments:

unsigned * *src*

unsigned * *dest*

unsigned *wordcnt*

Conditions:

This function requires a mongoose processor to be tested, it can not be tested on a development workstation.

Description:

Transfer `wordcnt` words indicated by `src` across the DMA interface to location indicated by `dest`. This operation can only be used between D-cache and memory, and memory to memory. If `src` and `dest` both point to D-cache, a panic will be generated.

34.6.23 FIOgetExpInfo()

#include fio.h

Scope: Science

Return type: **void**

Arguments:

unsigned **expnum*

unsigned **timestamp*

Conditions:

This function requires a mongoose processor to be tested, it can not be tested on a development workstation.

Description:

Return information about the current exposure, i.e., the exposure number and the timestamp of the exposure. Interrupts will be masked while this function executes. The interrupts will be disabled for less than ~5μseconds.

34.6.24 FIOgetNextCmd()

#include fio.h

Scope: Science

Return type: **bool**

Arguments

COMMAND **cmd*

Description:

Returns TRUE if a new science command has arrived, in which case, structure *cmd* contains the science command. Returns FALSE, and leaves *cmd* unchanged, if there are no new science commands. If the command is diagnostic in nature, this functions does the necessary processing before returning. Refer to the file "fio.h" for a list of possible diagnostic command types. All diagnostic command types are negative in value. Non-diagnostic command types, which are not defined in "fio.h", must have a value greater than 0.

34.6.25 FIOinit()

#include fio.h

Scope: Science

Return type: **void**

Arguments none

Description

This function initializes the various hardware registers in the FEP. The ring buffer is initialized by the BEP. Refer to the DPA Hardware Specification & System Description for a detailed accounting of the registers and initialization information.

34.6.26 FIOtouchWatchdog()

#include fio.h

Scope: Control

Return type: **void**

Arguments: none

Conditions:

This function requires actual ACIS hardware to be tested, it can not be tested on a development workstation.

Description

This function resets the watchdog timer to prevent a hardware reset.

34.6.27 FIOwriteCmdReply()

#include fio.h

Scope: Control

Return type: **void**

Arguments:

COMMAND* *cmdreply*

Description

This function writes the command into the command reply buffer. It is required after a call to **`FIOgetNextCmd()`** returns the value TRUE, and is also called with **`FIOgetNextCmd()`** to respond to diagnostic commands.

34.6.28 fioClearBitCtrlReg()

#include fio.h

Scope: Diagnostic

Return type **`void`**

Arguments:

unsigned *mask*

Conditions:

This function requires actual ACIS hardware to be tested, it can not be tested on a development workstation.

Description

This function clears the bits in the FEP Control Register indicated by the mask. It does not affect any of the other bits in the control register. The constants for setting particular bits are defined in the enum CtlRegMask in fio.h.

34.6.29 fioClearBitImCtrlReg()

#include fio.h

Scope: Diagnostic

Return type **`void`**

Arguments:

unsigned *mask*

Conditions:

This function requires actual ACIS hardware to be tested, it can not be tested on a development workstation.

Description

This function clears the bits in the FEP Image Control Register indicated by the mask. It does not affect any of the other bits in the register. The constants for setting particular bits are defined in the enum ImgCtlRegMask in fio.h.

34.6.30 fioGetCmd()

#include fio.h

Scope: Control

Return type: **void**

Arguments:

COMMAND * *cptr*

Conditions:

This function does not check if the command is current. Use **fioPollMBox()** to check the status of the command.

Documentation

Get a command from the BEP. The command is returned in the area indicated by *cptr*.

34.6.31 fioGetStatusReg()

#include fio.h

Scope: Diagnostic

Return type **unsigned**

Conditions:

This function requires actual ACIS hardware to be tested, it can not be tested on a development workstation.

Description

This function returns the value of the status register.

34.6.32 fioGetImStatusReg()

#include fio.h

Scope: Diagnostic

Return type **`unsigned`**

Conditions:

This function requires actual ACIS hardware to be tested, it can not be tested on a development workstation.

Description

This function returns the value of the Image Status register.

34.6.33 fioPollMBox()

#include fio.h

Scope: Control

Return type: **bool**

Arguments:

MBOXstate ******mbox*

MBOXstate *state*

Documentation

Returns TRUE if the MBOXstate of the mailbox matches *state* in the argument list. Returns FALSE otherwise. Mailbox state variables are defined in "fio.h".

34.6.34 fioRbStatus()

#include fio.h

Scope: Science

Return type:

RINGBUFstatus

Arguments:

unsigned *`available`

Description

This function returns the status of the ring buffer. This function should be called before any other functions concerning the ring buffer are used. Available contains the number of 32-word blocks available in the buffer.

Return Values

RB_FULL - Ring buffer is full, do not append more data

RB_EMPTY - Ring buffer is empty

RB_NOT_FULL - Ring buffer is not full and not empty, check *available* to see if there is enough room to append more data.

RB_ERROR - There is an internal error in the ring buffer

34.6.35 fioReadIcache()

#include fio.h

Scope: Diagnostic

Return type: **void**

Arguments:

unsigned * *src*

unsigned * *dest*

unsigned *wordcnt*

Conditions:

This function requires a mongoose processor to be tested, it can not be tested on a development workstation.

Description

This function reads *wordcnt* 32-bit words from the I-cache, starting at *src*, into the buffer indicated by *dest*.

34.6.36 fioReadMem()

#include fio.h

Scope: Diagnostic

Return type **void**

Arguments:

unsigned * *src*

unsigned * *dest*

unsigned *wordcnt*

Description

This function reads *wordcnt* 32-bit words starting at *src* in D-cache or general memory into the buffer indicated by *dest*.

34.6.37 fioSetBitCtrlReg()

#include fio.h

Scope: Diagnostic

Return type **void**

Arguments:

unsigned *mask*

Description

This function sets the bits in the Control register indicated by *mask*. It does not affect any of the other bits in the register. The constants for setting particular bits are defined in the enum CtrlRegMask in the file fio.h.

34.6.38 fioSetBitImCtrlReg()

#include fio.h

Scope: Diagnostic

Return type **void**

Arguments:

unsigned *mask*

Description

This function sets the bits in the Image Control register indicated by *mask*. It does not affect any of the other bits in the register. The constants for setting particular bits are defined in the enum ImgCtrlRegMask in the file fio.h.

34.6.39 fioSetSegAllocReg()

#include fio.h

Scope: Control

Return type **void**

Arguments:

SEGALLOC *segment*

Description

This function sets the bits in the Bulk Memory Segment Allocation register indicated by *segment*. It is used to designate where in bulk memory the parity plane, threshold plane, and overclock buffer are located. It does not affect any of the other bits in the register. The constants for setting particular bits are defined by the enum SEGALLOC in fio.h.

34.6.40 fioWriteIcache()

#include fio.h

Scope: Diagnostic

Return type: **void**

Arguments:

unsigned * *src*

unsigned * *dest*

unsigned *wordcnt*

Conditions:

This function requires a mongoose processor to be tested, it can not be tested on a development workstation.

Description

This function writes *wordcnt* 32-bit words starting at *src* into the I-cache starting at *dest*.

34.6.41 fioWriteMem()

#include fio.h

Scope: Diagnostic

Return type: **void**

Arguments:

unsigned * *src*

*unsigned * dest*

unsigned *wordcnt*

Description

This function writes *wordcnt* 32-bit words starting at *src*, into the D-cache or general memory starting at *dest*.

34.6.42 fioWritePulseReg()

#include fio.h

Scope: Diagnostic

Return type **void**

Arguments:

unsigned *mask*

Conditions:

This function requires actual ACIS hardware to be tested, it can not be tested on a development workstation.

Description

This function sets the bits in the Pulse register indicated by *mask*. It does not affect any of the other bits in the register. The constants for setting particular bits are by the enum PulseRegMask in fio.h.

34.6.43 fioWriteImPulseReg()

#include fio.h

Scope: Diagnostic

Return type **`void`**

Arguments:

unsigned *`mask`*

Conditions:

This function requires actual ACIS hardware to be tested, it can not be tested on a development workstation.

Description

This function sets the bits in the Image Pulse register indicated by *`mask`*. It does not affect any of the other bits in the register. The constants for setting particular bits are by the enum ImgPulseRegMask in fio.h.