CSR

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CENTER FOR SPACE RESEARCH
CAMBRIDGE, MASSACHUSETTS 02139

REVISION LOG	**TITLE: Software Detailed Design** **FEP Command Controller**	**DOC. NO.** **36-53236 Rev. A**

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
A	4/22/96	36-599	all	Initial released version. Incorporated comments from initial review.	5/23/96

41.0 FEP Command Controller (36-53236 A)

41.1 Purpose

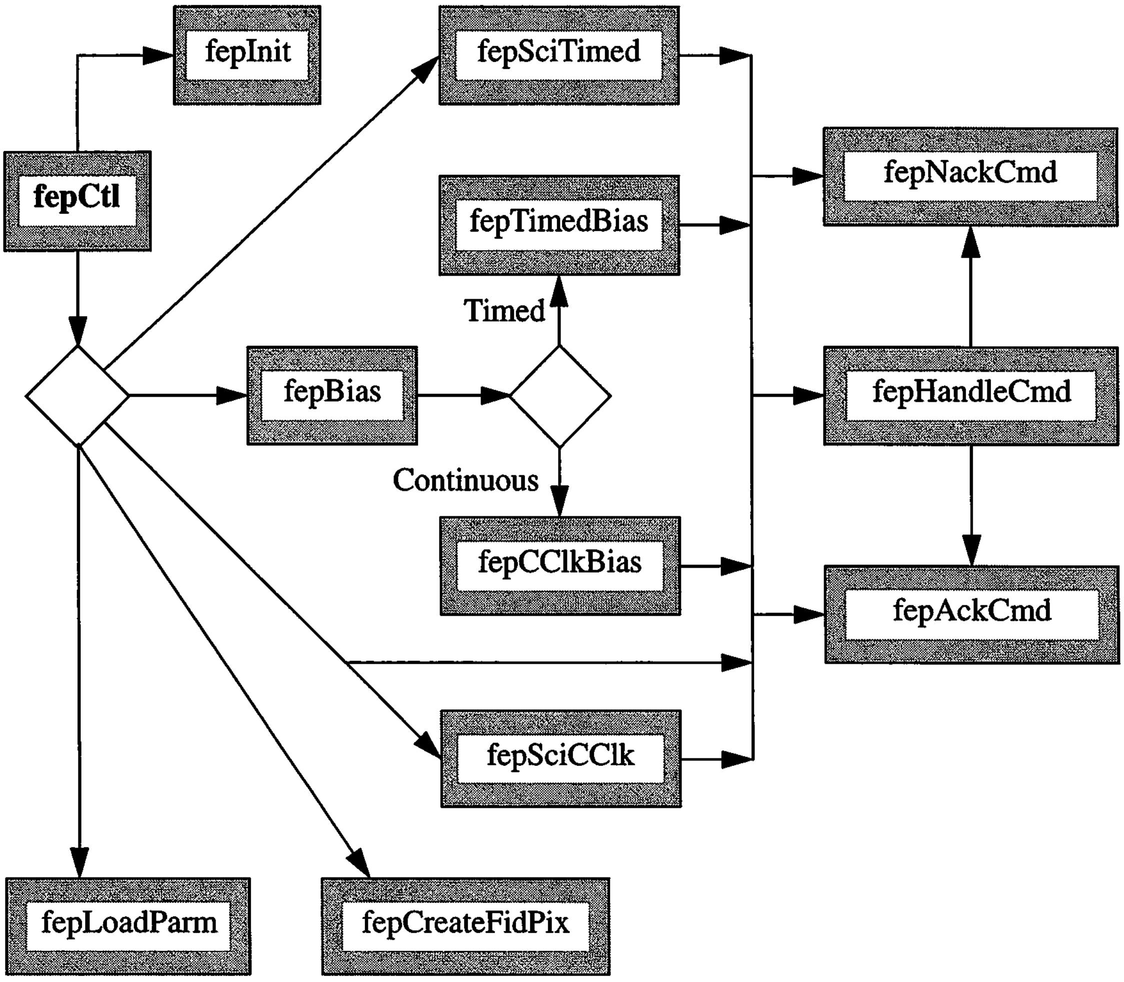
`fepCtl` performs high-level control functions within the FEPs, namely the interception of science commands from the BEP and their subsequent execution. Low level commands (memory read, write, and execute) are handled within the **FEP IO Library (36-53223 B)** routines (see Section 39.0) and are essentially transparent to the command controller and science modules.

41.2 Uses

The FEP Controller provides the following features:

Use 1:: Respond to a BEP science command from IDLE status
Use 2:: Respond to one BEP command while executing another

FIGURE 178. FEP controller subroutines and their calling hierarchy



41.3 Organization

The FEP Controller module contains the main FEP command processing loop, along with various external functions for frequently used FEP operations. Its stack contains a single copy of the `FEPparm` structure, whose address is passed to all FEP modules and functions. Thus if, for instance, `fepCtl` calls `fepSciTimed` to conduct a timed exposure science run, and `fepSciTimed` itself calls `fepHandleCmd` to process an incoming BEP command, it is `fepSciTimed`'s responsibility to ensure that it passes to `fepHandleCmd` the same pointer to `FEPparm` that it was itself given.

A brief description of the functions within this module is as follows:

- `fepCtl` - This is the top-level FEP processing loop. It allocates a single `FEPparm` structure on its stack and passes its address to all lower level science routines which can therefore use it as a substitute for `static` storage. By convention, this address parameter is called *`fp`* in all FEP modules. After calling `FIOinit` and `fepInit`, the code cycles endlessly over calls to `FIOgetNextCmd`. A `TRUE` return value signifies that a science-mode command has been received from the BEP, and the appropriate subroutine is called. The BEP itself will wait for the FEP to reply before sending another command. Finally, whether or not a command is processed, a call to `FIOtouchWatchdog` resets the watchdog timer, and the loop repeats.
- `fepAckCmd` - This function is called to respond positively to a BEP science command. It is global, since it can be called from other FEP science modules.
- `fepAppendRingBuf` - This function is called to add a block of data to the ring buffer, from whence it will be read by the BEP. The routine is global since it is only called from other FEP science modules.
- `fepBias` - This function is called when a bias calibration command is received from the BEP. It inspects the `FEPbiasRec` *`type`* code in *`fp->tp`*, and calls the appropriate bias function.
- `fepCreateFidPix` - This function is called when a `BEP_FEP_CMD_FIDPIX` command is received from the FEP. It removes any previously defined fiducial pixels, saves the addresses of new ones in *`fp->fidpix`* and purposely sets the corresponding bits of the bias parity plane to incorrect values, *i.e.* 0 becomes 1 and 1 becomes 0, in order to force the FEP to examine these pixels after every exposure.
- `fepEnableNextFrame` - This function is called to tell the FEP hardware thresholder to start storing image pixels when it next receives a `VSYNC` signal. It is global since it is only called from other FEP science modules.
- `fepHandleCmd` - This function is called within the `fepCtl` loop, and may also be called from any science function, when a positive return code from a call to `FIOgetNextCmd` indicates that the BEP has issued another science command.
- `fepInit` - This function is called once at start-up time. It initializes the *`flags`* field and the `FEPbiasRec`, `FEPstatus`, `FEPexpRec`, and `FEPexpEndRec` structures within the `FEPparm` structure.

- fepLoadParm - This function is called when a parameter block is received from the BEP via the BEP-FEP command mailbox. It copies the block to *fp->tp*, performs a series of validation tests, and then calls either fepAckCmd (or fepNackCmd) to tell the BEP that the block was received and that it passed (or failed) the validation.
- fepNackCmd - This function is called to respond negatively to a BEP science command, or when the command code returned by the FIOgetNextCmd call is not recognized. It is global, since it can be called from other FEP science modules.
- fepSetAddrMode - This function is called when initializing the various command modes to set the FEP hardware registers according to the desired DEA output node configuration. The routine is global since it is only called from other FEP science modules.

41.4 Global Variables

Table 39 lists those FEPparm fields that are referenced within the FEP command controller module. They are defined in *fepCtl.h* and always addressed by a pointer parameter named *fp*:

TABLE 39. Global FEPparm fields used by the Command Controller

FEPparm Substructure	Variable Name	Description
bepCmd		Latest command received from BEP, and the FEP's reply
	args	Command/reply contents
	len	Command/reply length in fullwords
	type	Command/reply type code (see *fepBep.h*)
br		Bias calibration parameters
	bias0[4]	Average overclock values of each output node for the first data frame used for bias calibration
	biassum	Checksum of bias0 values, initialized to 0xffffffff unless valid bias parameters are extracted from I-cache
ex		FEPexpRec exposure record
	expnum	Current exposure frame
	timestamp	Current frame's arrival time
	type	Initialized to FEP_EXPOSURE_REC
exend		FEPexpEndRec end-of-exposure record
	type	Initialized to FEP_EXPOSURE_END_REC
fepStatus		FEP status reported to BEP
	biasflag	TRUE if a valid bias map exists, else FALSE
	mode	Initialized to zero
	parityplane	Address of the start of the bias parity plane
	bias0[4]	Initialized to *bias0[4]*, if bias valid, otherwise zeroes
flags		Flag bits defined in *fepCtl.h*
	FP_SUSPEND	BEP has sent BEP_FEP_CMD_SUSPEND
	FP_TERMINATE	BEP has sent BEP_FEP_CMD_STOP
tp		Exposure parameter block
	btype	Initialized to FEP_NO_BIAS (see *fepBep.h*)
	ncols	Number of pixels per output node
	quadcode	Output node clocking mode (see *fepBep.h*)
	type	Initialized to FEP_NO_PARM (see *fepBep.h*)
fidpix[MAX_FID_PIX]		Row and column addresses of fiducial pixels.
nfidpix		Number of fiducial pixels to be reported in timed-exposure event detecting modes, set by the most recent call to fepCreateFidPix.
nextexpnum		Next possible frame number to be processed, set by a call to fepEnableNextFrame

41.5 Scenarios

41.5.1 Use 1: Respond to a BEP science command from IDLE status

The `fepCtl` loop receives a positive return code from a call to `FIOgetNextCmd`, signifying that a command has been received from the BEP and is sitting in the BEP-to-FEP mailbox. The command *`type`* is copied to *`fp->fepStatus.mode`* and examined to determine the action to be taken, as shown in Table 40.

Parameter loads (BEP_FEP_CMD_PARAM) and status requests (BEP_FEP_CMD_STATUS) are handled entirely within the ***fepCtl*** module and have no immediate external consequence. Commands to start science runs (BEP_FEP_CMD_TIMED and BEP_FEP_CMD_CCLK) and bias calibrations (BEP_FEP_CMD_BIAS) cause the corresponding science modules to be invoked. The remaining commands—BEP_FEP_CMD_STOP to terminate a running command, and BEP_FEP_CMD_SUSPEND and BEP_FEP_CMD_RESUME to temporarily suspend and restart a command—have no meaning since no command is running. `fepCtl` therefore responds by calling `fepNackCmd` to indicate to the BEP that this was the case.

Note that if ***fepCtl*** calls another module to execute the command, it is the responsibility of the called function to make a prompt call to either `fepAckCmd` or `fepNackCmd` to tell the BEP whether the command was accepted or rejected.

TABLE 40. FEP responses to BEP commands received in IDLE mode

Value of fp->bepCmd.type*	Function Called	Description
BEP_FEP_CMD_BIAS	`fepBias`	Start a bias calibration
BEP_FEP_CMD_CCLK	`fepSciCClk`	Start a continuously clocked science run
BEP_FEP_CMD_FIDPIX	`fepCreateFidPix`	Load zero or more fiducial pixel addresses into *`fp->fidpix`*.
BEP_FEP_CMD_PARAM	`fepLoadParm`	Load and check a new FEP parameter block
BEP_FEP_CMD_RESUME	`fepNackCmd`	Signal the BEP that the command is inappropriate in IDLE mode
BEP_FEP_CMD_STATUS	`fepHandleCmd`	Return the current FEP software status
BEP_FEP_CMD_STOP	`fepNackCmd`	Signal the BEP that the command is inappropriate in IDLE mode
BEP_FEP_CMD_SUSPEND	`fepNackCmd`	Signal the BEP that the command is inappropriate in IDLE mode
BEP_FEP_CMD_TIMED	`fepSciTimed`	Start a timed exposure science run
None of the above	`fepNackCmd`	Signal the BEP that the command is not understood

* see the `#define` statements in *fepBep.h*.

41.5.2 Use 2: Respond to one BEP command while executing another

The "high-level" science processing routines—fepSciTimed, fepSciCClk, and their corresponding bias calculation routines, fepTimedBias, and fepCClkBias—should make frequent calls to FIOgetNextCmd to determine whether a command has been received from the BEP. When this function returns TRUE, it indicates that a high-level command has been received via the BEP-FEP mailbox.[1] The science routine should immediately call fepHandleCmd, which uses the value of *fp->bepCmd.type* to determine the action to take, as shown in Table 41:

TABLE 41. FEP responses to BEP commands received during a science or bias run

Value of fp->bepCmd.type	Function Called	Description
BEP_FEP_CMD_BIAS	fepNackCmd	This command is unanticipated while a science or bias run is in progress.
BEP_FEP_CMD_CCLK	fepNackCmd	This command is unanticipated while a science or bias run is in progress.
BEP_FEP_CMD_FIDPIX	fepNackCmd	This command is unanticipated while a science or bias run is in progress.
BEP_FEP_CMD_PARAM	fepNackCmd	This command is unanticipated while a science or bias run is in progress.
BEP_FEP_CMD_RESUME	fepAckCmd	Signal that processing is to be resumed by clearing the FP_SUSPEND bit in fp->flags.
BEP_FEP_CMD_STATUS	FIOwriteCmdReply	Call FIOwriteCmdReply directly to reply to the BEP, passing as argument the current *fp->fepStatus* structure.
BEP_FEP_CMD_STOP	fepAckCmd	Signal that the run is to be terminated by setting the FP_TERMINATE bit in fp->flags.
BEP_FEP_CMD_SUSPEND	fepAckCmd	Signal that processing is to be temporarily suspended by setting the FP_SUSPEND bit in *fp->flags*. The suspension is performed entirely by the science process.
BEP_FEP_CMD_TIMED	fepNackCmd	This command is unanticipated while a science or bias run is in progress.
Any other value	fepNackCmd	-

[1] Low-level commands from the BEP are handled transparently within FIOgetNextCmd and are never reported to the high-level science layer.

41.6 Specification

41.6.1 fepCtl()

Scope: Science.

Return Type: void.

Arguments: none.

Description:

Once the bootServerFep loader has copied the FEP executable image to I-Cache memory, the BEP sends it a CMD_EXECUTE_MEM command to branch to the fepCtl entry point. This function starts out by calling FIOinit to initialize the low-level library functions, and fepInit to initialize fields in fepCtl's automatic FEPparm structure, as shown in Section 41.4.

fepCtl then executes an endless loop, alternately calling FIOgetNextCmd to see whether a command has arrived from the BEP, and FIOtouchWatchdog to reset the watchdog timer. Whenever FIOgetNextCmd returns a positive value, fepCtl inspects the *bepCmd.type* code and calls the appropriate function (see Table 40).

fepCtl never returns. This is a *good thing*, since it is either called directly from the FEP loader executing in bulk memory, or through a small assembler-language stub, and the return path will almost certainly have been overwritten long since.

41.6.2 fepAckCmd()

Scope: Science.

Return Type: void.

Arguments:

*FEPparm *fp*

Description:

A command previously received from the FEP is positively acknowledged by (a) setting the reply type to the *fp->bepCmd. type* of the original command, (b) setting the reply length to 2, (c) setting the single reply value to TRUE, and (d) calling `FIOwriteCmdReply` to send the reply back to the BEP.

41.6.3 fepAppendRingBuf()

Scope: Science.

Return Type: void.

Arguments:.

*unsigned *ptr*

unsigned wordcnt

*FEPparm *fp*

Description:.

fepAppendRingBuf calls FIOappendBlock to write *ptr* to the ring buffer in segments of no more than 32 words each. If FIOappendBlock returns FALSE, it indicates that a command has been received from the BEP, and FIOgetNextCmd will be called to process it. If FIOgetNextCmd returns TRUE, this was a high-level command and fepHandleCmd is then called to give it further processing. Once any command is processed, fepAppendRingBuf resumes its calls to FIOappendBlock, not returning until all *wordcnt* words have been copied to the ring buffer.

41.6.4 fepBias()

Scope: Static.

Return Type: void.

Arguments:

*FEPparm *fp*

Description:

The `fp->tp.type` and `fp->tp.btype` fields are inspected and the appropriate bias calibration routine called, as shown in Table 42.

The bias routine, `fepTimedBias` or `fepCClkBias`, must immediately inspect the parameter block, `fp->tp`, for validity and respond `fepAckCmd` or `fepNackCmd`, as appropriate, so that the BEP can be assured that the bias command has been received.

TABLE 42. Selection of Bias Calibration Function

fp->tp.type value	fp->tp.btype value	Function called	Comments
FEP_TIMED_PARM_RAW FEP_TIMED_PARM_HIST FEP_TIMED_PARM_3x3 FEP_TIMED_PARM_5x5	FEP_BIAS_1 FEP_BIAS_2	`fepTimedBias`	Start a timed-exposure bias calibration
	FEP_NO_BIAS	`fepAckCmd`	No bias calibration required
	Other	`fepNackCmd`	Unexpected `btype` value
FEP_CCLK_PARM_RAW FEP_CCLK_PARM_1x3	FEP_BIAS_1 FEP_BIAS_2	`fepCClkBias`	Start a continuously clocked bias calibration
	FEP_NO_BIAS	`fepAckCmd`	No bias calibration required
	Other	`fepNackCmd`	Unexpected `btype` value
Any other value, including FEP_NO_PARM	Any	`fepNackCmd`	Unexpected `type` value

41.6.5 fepCreateFidPix()

Scope: Science.

Return Type: void.

Arguments:

*FEPparm *fp*

Description:

This is called when fepCtl receives a BEP_FEP_CMD_FIDPIX command while in *IDLE* mode. It checks the *fp->bepCmd.len* field for legality[1], returning FEP_CMD_ERR_PARM_LEN to the BEP if illegal. It then determines whether the bias map is valid (is *fp->fepStatus.biasflag true*?). If not, it returns FEP_CMD_ERR_NO_BIAS to the BEP.

If the tests succeed, any existing fiducial pixels are cleared out of the bias parity plane (see below) and the stored count *fp->nfidpix* is set to zero. Then the first *fp->bepCmd.len*-1 elements of *fp->bepCmd.args* are accepted as defining the new fiducial pixel list for subsequent time exposure science runs using the current bias map. Within each 32-bit element, the 12 low order bits (0-11) define the column[2] and bits 16-27 define the row. For each new fiducial pixel, the corresponding bias parity plane bit is set to an *incorrect* value, relative to its 12-bit value in the bias map, thereby causing the hardware thresholder to flag it as a parity error on every exposure frame. This in turn will allow the timed-exposure event-detection routine (FEPtestEvenPixel) to recognize and store it as a fiducial pixel.

When the fiducial pixel address list has been processed, fepCreateFidPix calls fepAckCmd to pass a FEP_CMD_NOERR return code to the BEP.

[1] 1 ≤ *fp->bepCmd.len* ≤ MAX_FID_PIX.

[2] Since fiducial pixels are always reported in contiguous even/odd pairs, an odd column address will be decremented upon receipt.

41.6.6 fepEnableNextFrame()

Scope: Science.

Return Type: void.

Arguments:.

*FEPparm *fp*

Description:.

This function is called to signal the FEP hardware to store then next image frame, i.e. the pixels that follow the next VSYNC code. This is done with a call to fioWriteImPulseReg(IPULSE_ARMNXTACQ), sandwiched between calls to FIOgetExpInfo. If the exposure number is found to have changed between these calls, it implies that a VSYNC code was encountered. Since the software has no way of knowing whether the VSYNC was received before or after the pulse register was updated, the call to fioWriteImPulseReg will be issued a second time, causing a frame to be skipped.

On exit, fp->ex.expnum contains the current value of the exposure counter, fp->ex.timestamp contains the corresponding latched clock time, and fp->nextexpnum = fp->ex.expnum + 1.

41.6.7 fepHandleCmd()

Scope: Science.

Return Type: void.

Arguments:

*FEPparm *fp*

Description:

This function is called whenever a call to fepGetNextCmd returns TRUE, indicating that a BEP command has been received. The responses are determined by the *bepCmd.type* value and are listed in Table 41 on page 1197. In each case, a reply is sent to the BEP: usually either positive acknowledgment (fepAckCmd) or negative acknowledgment (fepNackCmd), or, in response to a BEP_FEP_CMD_STATUS request, FIOwriteCmdReply is called to return the contents of *fp->fepStatus* in the reply argument list.

41.6.8 fepInit()

Scope: Static.

Return Type: void.

Arguments: none.

Description:

This function is called to initialize the Front End Processor's FEPparm parameter block, as shown in Table 43. Bias values are restored from I-cache by a call to FIOgetBiasConfig. If br.biassum is consistent with the bias0 array, it is assumed that the bias map itself is still usable, so fepStatus.biasflag is set TRUE. Otherwise, it is set FALSE and br.biassum is initialized to the "impossible" value 0xffffffff.

TABLE 43. FEPparm variables Initialized by fepInit

Variable	Initial Value
br.biassum	0xffffffff unless the value copied from I-cache was consistent with the values of *br.bias0*[] also copied from I-cache.
ex.type	FEP_EXPOSURE_REC
exend.type	FEP_EXPOSURE_END_REC
fepStatus.mode	0
fepStatus.biasflag	TRUE if *br.biassum* is valid, otherwise FALSE.
fepStatus.parityplane	FIOgetBiasParityPlanePtr() value
fepStatus.bias0[]	*br.bias0*[] if *br.biassum* is valid, otherwise zeroes.
flags	0
nfidpix	0
tp.type	FEP_NO_PARM

41.6.9 fepLoadParm()

Scope: Static.

Return Type: void.

Arguments:

*FEPparm *fp*

Description:

This function copies a FEP parameter block from the command mailbox *fp->bepCmd.args* to *fp->tp*. It checks the parameters shown in Table 44 and calls fepAckCmd if they are valid, or fepNackCmd if they are not.

TABLE 44. FEPparm variables that are tested by fepLoadParm

Variable Name	Validity Test
btype	Any legal fepBiasType value
ncols	Even and within the range 2-256
noclk	Even and within the range 0-32
nrows	Within the range 1-1024
quadcode	legal fepQuadCode value
type	legal fepParmType value

41.6.10 fepNackCmd()

Scope: Science.

Return Type: void.

Arguments:

*FEPparm *fp*

fepCmdRetCode errno

Description:

A command previously received from the FEP is negatively acknowledged by (a) setting the reply type to the *fp->bepCmd. type* of the original command, (b) setting the reply length to 2, (c) setting the single reply value to *errno*, and (d) calling FIOwriteCmdReply to send the reply back to the BEP. *errno* values are tabulated in *fepBep.h.*

41.6.11 fepSetAddrMode()

Scope: Science.

Return Type: void.

Arguments:.

fepQuadCode quadcode

unsigned ncols

bool tHold

Description:

This function loads FEP hardware registers with values derived from *quadcode*, the current DEA clocking mode, and *ncols*, the number of pixels per output node per row, as shown in Table 45. It calls `fioClearBitCtrlReg` and `fioSetBitCtrlReg` to set the control register, and `FIOsetOffsetReg` to set the four address offset registers.

FEP science and bias modules call the `FIOsetAddrMode` function to set FEP hardware registers so that the FEP addressing hardware can correctly interpret the incoming image pixels. It does this on the basis of the *quadcode* and *ncols* parameters, the former to specify the DEA output node configuration, and the latter to show the number of pixels to be read from a CCD quadrant by each output node.

The *tHold* parameter determines whether FEP hardware thresholding and bias parity detection should be turned on (`TRUE`) or off (`FALSE`). Overclock processing is always enabled.

The result will be to initialize the FEP image control register, and the four address offset registers, as shown in Table 45. Note that the fourth (diagnostic) output clocking possibility, in which nodes A–D are clocked "backwards", need not be distinguished from FEP_QUAD_ABCD within the FEP, and the latter is therefore used for both.

TABLE 45. FEP hardware register configuration for CCD output clocking modes

Value of quadcode	FEP Img Control Register	FEP Address Offset Registers			
		Offset$_0$	Offset$_1$	Offset$_2$	Offset$_3$
FEP_QUAD_ABCD	0x2a	0	2**ncols*–1	2**ncols*	4**ncols*–1
FEP_QUAD_AC FEP_QUAD_BD	0x1a	0	2**ncols*–1	2**ncols*	4**ncols*–1