CSR

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

REVISION LOG

TITLE: Software Detailed Design
FEP Science Timed Exposure Mode

DOC. NO.
36-53224 Rev. B

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
-	3/30/95	-	all	New document	
A	5/11/95	36-254	all	Incorporate review comments	RFG 5/22/95
B	6/17/96	36-623	1213, 1214	Added support to ignore "n" initial frames	6/14/96

18.0 FEP Timed Exposure Modes (36-53224 B)

18.1 Purpose

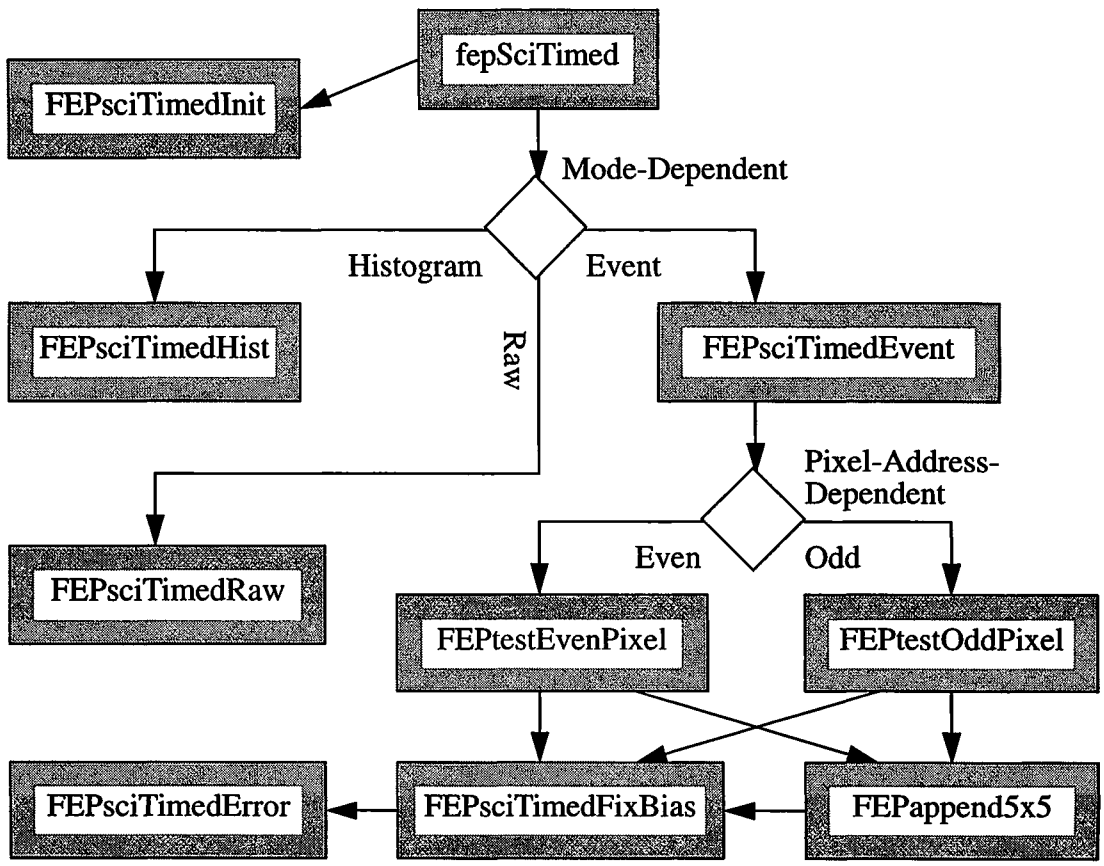
The ***fepSciTimed*** module, executing in the FEP, implements timed-exposure science processing, comprising either raw mode, event detection mode, or raw histogramming mode. It is called from `fepCtl` with a single argument, *fp*, a pointer to the `fepParm` structure. Before invoking `fepSciTimed`, the FEP must be commanded to (a) load a timed-exposure `FEPparmBlock` into *fp->tp*, and (b) perform a bias calibration.

18.2 Uses

The ***fepSciTimed*** function operates in one of the following modes, according to the value of *fp->tp.type*:

Use 1:: Identify and report 3x3 candidate X-ray events to the BEP.
Use 2:: Identify and report 5x5 candidate X-ray events to the BEP.
Use 3:: Report rows of raw pixels to the BEP.
Use 4:: Accumulate raw pixel histograms and report them to the BEP.

FIGURE 179. fepSciTimed subroutines and their calling hierarchy

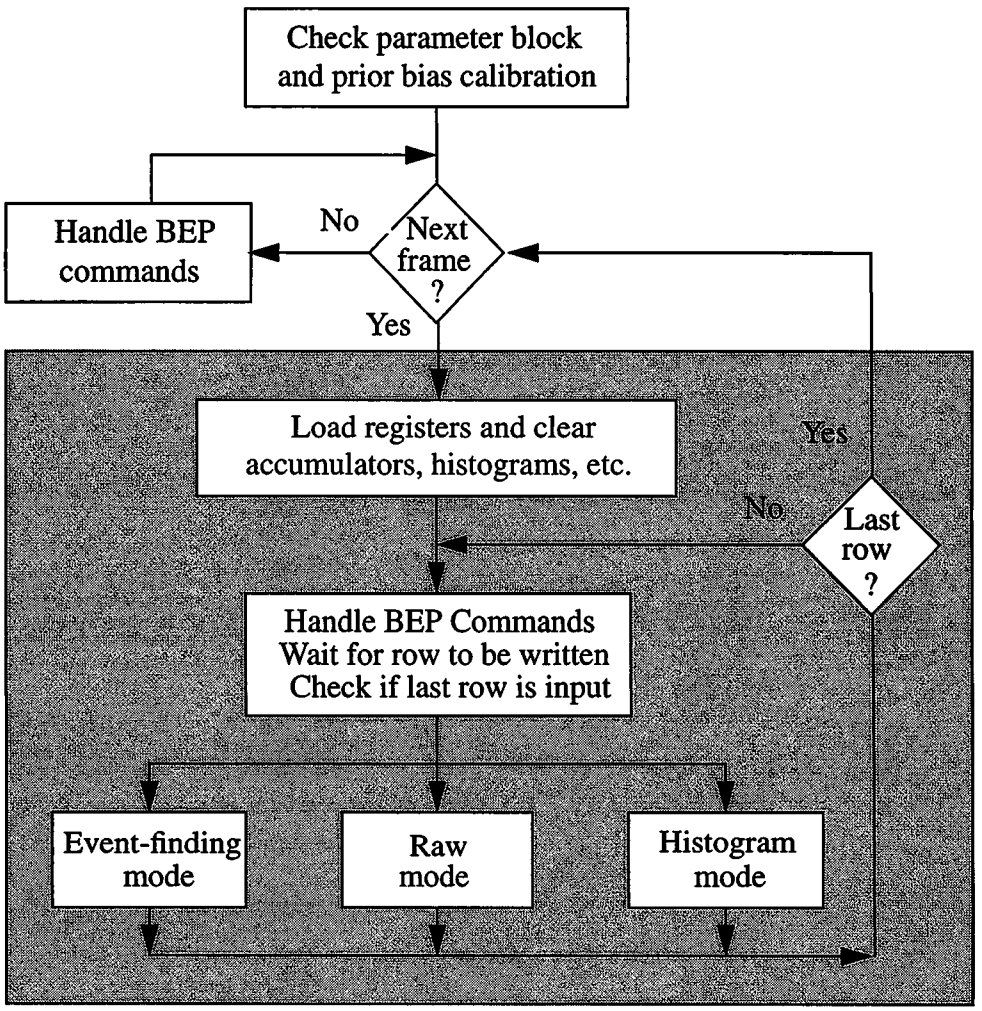


18.3 Organization

All interactions with the BEP and with FEP hardware are made through ***fepio*** library functions described in Section 39.0. The data structures that are passed between FEP and BEP are defined in that document and in Section 4.10. `fepSciTimed` makes use of the following functions which call each other in the manner shown in Figure 179:

- `FEPsciTimedInit`—validates the `FEPparmBlock`, *`fp->tp`*, and performs all necessary mode-dependent initialization, e.g. in event-finding modes, it checks that the bias map has been initialized.
- `FEPsciTimedEvent`—processes a single timed-exposure frame in event-finding mode, calling `FEPtestEvenPixel` and `FEPtestOddPixel` for each detected threshold crossing.
- `FEPsciTimedHist`—processes a single timed-exposure frame in raw histogram mode, reporting pixel frequency histograms and overclock statistics in `FEPeventRecHist` records.

FIGURE 180. The flowchart of fepSciTimed



- `FEPsciTimedRaw`—processes a single timed-exposure frame in raw mode, reporting raw pixels and overclocks in `FEPeventRecRaw` records without any `thresholdingFEPtestEvenPixel`—processes a threshold-crossing event possessing an even bit-offset relative to the start of the T-Plane buffer. If the center pixel is a local maximum, `FIOappendData` is called to copy a `FEPeventRec3x3` record (or, in 5x5 mode, a `FEPeventRec5x5` record) to the ring buffer.
- `FEPtestOddPixel` —processes a threshold-crossing event possessing an odd bit-offset relative to the start of the T-Plane buffer.If the center pixel is a local maximum, `FIOappendData` is called to copy a `FEPeventRec3x3` record (or, in 5x5 mode, a `FEPeventRec5x5` record) to the ring buffer.
- `FEPsciPixTest`—is an `inline` function (and therefore not shown in Figure 179) that is called several times within `FEPtestEvenPixel` and `FEPtestOddPixel` to determine whether the central pixel is less than (or equal to) one of its neighbors.
- `FEPappend5x5`—is called in 5x5 event-finding mode to (a) append the 16 edge pixels and bias values surrounding a core 3x3 event to a `FEPeventRec3x3` structure before it is copied to the ring buffer by `FEPtestEvenPixel` or `FEPtestOddPixel`, and (b) change its `type` to `FEP_EVENT_REC_5x5`, thereby transforming it into a `FEPeventRec5x5` structure.
- `FEPsciTimedFixBias`—is called if a parity error is discovered in one or both of a pair of bias map values. It calls `FEPsciTimedError` to reset the parity plane and T-plane bits, calls `FIOappendData` to copy a `FEPerrorRec` record to the ring buffer, and returns to the caller the corrected value of the bias map pair.
- `FEPsciTimedError` —resets the appropriate bits in the parity and threshold planes.

18.4 Global Variables

The following FEPparm fields, defined in *fepCtl.h* and invariably addressed by the fp pointer parameter, are used by all timed exposure modes:

bepCmd	latest command received from BEP
ex	current FEPexpRec record
exend	current FEPexpEndRec record
flags	flag bits:
FP_EDGE_ROW	current row is at the top or bottom of the frame
FP_SUSPEND	BEP has sent BEP_FEP_CMD_SUSPEND
FP_PAST_EOR	FEP hardware has finished with the current frame
FP_TERMINATE	BEP has sent BEP_FEP_CMD_STOP
nextexpnum	the next exposure index that the FEP is to process
quadrants	the number of DEA output nodes being sampled
tp	exposure parameter block
initskip	number of initial frames to ignore
ncols	number of CCD columns per output node
noclk	number of overclocks per output node
nrows	number of CCD rows between frame markers
quadcode	output node clocking mode
thresh[4]	threshold values for each output node
type	processing mode, either FEP_CCLK_PARM_1x3 or FEP_CCLK_PARM_RAW.

The following FEPparm fields are only used by fepSciTimedEvent:

br	pointer to bias calibration parameters
colshft	bit shift to transform column index to node index
fidpix[]	fiducial pixel address list
image	pointer to start of current image row
lastcol	index of last image column
nfidpix	fiducial pixel count

The following FEPparm fields are only used by fepSciTimedHist:

expcount	count of processed exposures
phist	pointer to FEPeventRecHist structure

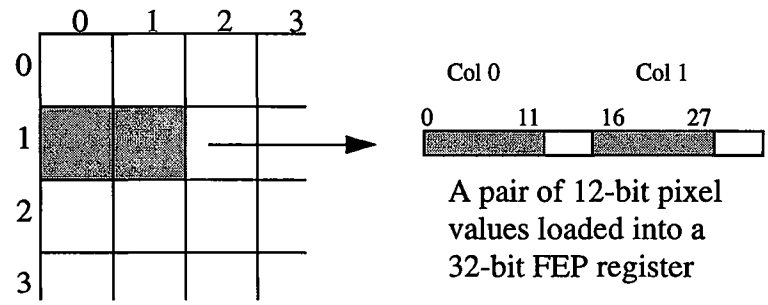
18.5 Scenarios

The following paragraphs describe the basic functions performed in timed exposure mode. All Timed Exposure modes wait until an image frame is processed and copied to the ring buffer before commanding the hardware thresholder to read the next frame, thereby ensuring that no partially processed frames are generated.

18.5.1 Use 1: Report 3x3 Events

- This mode is characterized by *fp->tp.type* == FEP_TIMED_PARM_3x3. After checking that the bias array contains appropriate values, FEPsciTimedInit calls fioWriteImPulseReg to start the hardware thresholder, and waits for frames to arrive. The first *fp->tp.initskip* frames are ignored.
- Once a valid frame arrives, fepSciTimed copies the exposure number and frame arrival time into a FEPexpRec structure and calls FIOappendData to copy it to the ring buffer. It then calls FEPsciTimedEvent to process the frame.
- FEPsciTimedEvent examines input rows one at a time. First, it calls FIOgetNextCmd to handle any incoming BEP commands, and then calls FIOgetExpInfo and/or FIOgetImageMapRowPtr until the hardware signals that the next row has been read into the image map.
- FEPsciTimedEvent then accumulates the current row's overclocks, adding them to the appropriate elements of *oSum*[], indexed by their DEA output nodes. It then inspects the T-plane buffer and calls either FEPtestEvenPixel or FEPtestOddPixel whenever a non-zero bit is found, indicating that the thresholder has located a pixel that exceeds its bias value by *fp->tp.thresh*[nn], where nn is the index of the appropriate DEA output node. Since the 12-bit pixel and bias values are only accessible two at a time via 32-bit CPU instructions, the logic required to inspect even-indexed pixels differs considerably from that needed for odd-indexed pixels, hence the two separate routines.

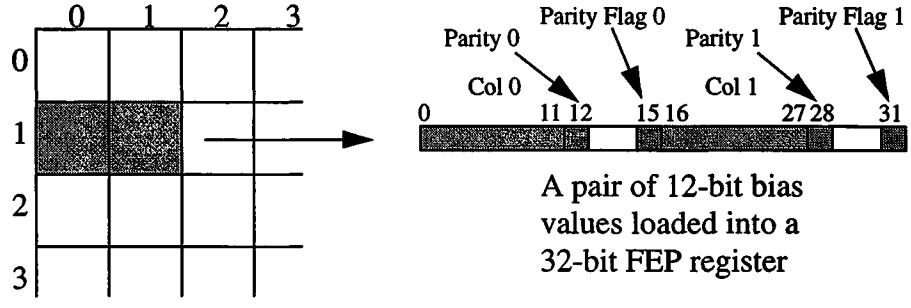
FIGURE 181. The Relation Between Image Pixels and FEP Register Values.



- The two pixel-testing routines examine the 8 pixels that surround the pixel that triggered the hardware thresholder, as illustrated in Figure 181. Pixels in the bad pixel list (with a bias value of 4095) are ignored, along with those whose bias values have been damaged since they were last calibrated, and were therefore set to 4094 during a sci-

ence run. If any of the remainder exceed the central pixel in relative (pixel–bias) value, or if any of the 4 pixels that follow the central pixel have the same relative value, the event is discarded; otherwise, `FIOappendData` is called to copy the 9 pixel and bias values to the ring buffer.

FIGURE 182. The Relation Between the Bias Map and FEP Register Values.



- Whenever `FEPtestEvenPixel` or `FEPtestOddPixel` loads a pair of bias values, as illustrated in Figure 182, it inspects their parity error flags (bits 15 and 31). If either is set, indicating that the parity of the bias value doesn't match the value of the corresponding bit in the bias parity plane, `FEPsciTimedFixBias` is called to reset the bias parity bit and the T-plane bit, and to return the "fixed" bias value (4094). The caller then stores the corrected value back into the bias map.
- Fiducial pixels are treated as a special case of bias parity errors, i.e. after the bias map and its parity plane are created, the BEP sends a `BEP_FEP_CMD_FIDPIX` command to the FEP, which calls the `fepCreateFidPix` routine (in ***fepCtl***). This sets the corresponding parity plane bits to *incorrect* values, which will cause the FEP thresholder to mark them in the T-plane. They will therefore always generate calls to `FEPtestEvenPixel` and it is the responsibility of `FEPsciTimedFixBias` to determine that they do not represent a genuine parity error, but to report them to the BEP by appending `FEPfidPixRec` records to the ring buffer.
- After processing the last row of pixels, `FEPsciTimedEvent` normalizes the *oSum*[ ] values and calls `FIOsetThresholdRegister` to update the hardware threshold registers.
- Finally, `FIOappendData` is called to copy a `FEPexpEndRec` (end-of-frame) record to the ring buffer.

18.5.2 Use 2: Report 5x5 Events

- This mode is characterized by *fp->tp.type* == `FEP_TIMED_PARM_5x5`. It is identical to the 3x3 mode described in the preceding section except that, prior to copying an event record to the ring buffer, `FEPtestEvenPixel` and `FEPtestOddPixel` call the `FEPappend5x5` function to transform the `FEPeventRec3x3` record into a `FEPeventRec5x5` record by setting its `type` to `FEP_EVENT_REC_5x5` and adding the 16 additional edge pixels and bias values.

18.5.3 Use 3: Report Raw Pixels

- This mode is characterized by *fp->tp.type* == FEP_TIMED_PARM_RAW. It begins in an identical manner to the event-finding modes described in the previous sections, except that fepSciTimed calls FEPsciTimedRaw to process each frame.
- FEPsciTimedRaw determines whether it can process the remainder of a current frame before it is overwritten by comparing the per-row processing time and data volume with the available time and ring-buffer space.
- FEPsciTimedRaw copies each row of pixels, and up to 30 overclocks from each DEA output node, to a FEPeventRecRaw record and thence to the ring buffer.

18.5.4 Use 4: Report Raw Pixel Histograms.

- This mode is characterized by *fp->tp.type* == FEP_TIMED_PARM_HIST. Processing commences in the manner described in the preceding sections, except that fepSciTimed calls FEPsciTimedHist to process each frame.
- FEPsciTimedHist adds each row's pixels to the histogram arrays, one per DEA output node.
- Each row's overclocks are also segregated by output node (*iquad*) and their cumulative minimum, maximum, sum, and sum-of-squares are saved, respectively, in the *iquad* elements of the *fp->fhist->omin*, *fp->fhist->omax*, *osum*, *ossqh*, and *ossql* arrays. Since the sum-of-squares may overflow an integer register, 64-bit arithmetic is used, i.e. the sum is stored in two 32-bit fields as (*ossqh*[*iquad*] + BIGNUM**ossql*[*iquad*]), where BIGNUM, a constant defined in *fepCtl.h*, does not exceed 2^{32}-1.
- After processing each frame, the mean overclock value and its variance are computed for each output node. The mean value is simply the *osum* value divided by the number of overclocks summed, *nn*, i.e. in integer arithmetic, i.e., (*osum*[*nquad*]+*nn*/2)/*nn*. Computation of the variance is complicated by the need to preserve round-off accuracy. It is formally (*ossql*[*iquad*] +BIGNUM**ossqh*[*iquad*])/*nn*-(*osum*[*iquad*]/*nn*)*(*osum*[*iquad*]/*nn*). The calculation is broken down into the following steps:

  div = *osum*[*iquad*] / *nn*;

  rem = *osum*[*iquad*] % *nn*;

  variance = *ossqh*[*iquad*] * (BIGINT / *nn*) - *div* * *div*
  + (*ossqh*[*iquad*] * (BIGINT % *nn*) + *ossql*[*iquad*]
  - 2 * *div* * *rem* - (*rem* * *rem* + *nn*/2)/*nn* + *nn*/2)/*nn*;

- The mean overclock and variance values are then added to the *fp->fhist->omean* and *fp->fhist->ovar* arrays.
- After processing the last row of a series of *fp->tp.nhist* exposures, FEPsciTimedHist divides *fp->fhist->omean* and *fp->fhist->ovar* by *fp->tp.nhist*, and calls FIOappendData to copy the results, along with the pixel histograms themselves, to the ring buffer in a FEPeventRecHist structure.

18.6 Specification

This section describes the functions that are local to the `fepSciTimed` unit. The only external is `fepSciTimed` itself which is called from `fepCtl`. The `FEPparm` structure is defined in *fepCtl.h*, along with several pixel access macros. The interface to the FEP I/O library is described in Section 39.0, and data and messages exchanged between FEP and BEP are described in Section 4.10.

18.6.1 fepSciTimed()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPparm *fp*

Description:

`fepSciTimed` is called from `fepCtl` with a single argument: the pointer `fp` to the `fepParm` structure. It is responsible for all FEP actions associated with a timed exposure science run, except for bias calibration. It performs the following actions:

- Calls `FEPsciTimedInit` to validate the BEP's parameter block, *fp->tp*, for the current mode, to verify that pixel biases have been computed, and to initialize mode-dependent variables. If `FEPsciTimedInit` returns `FEP_CMD_NOERR`, `fepSciTimed` calls `fepAckCmd` and continues. Otherwise, it calls `fepNackCmd` to pass the error code to the BEP, indicating that the command has failed, and `fepSciTimed` then returns to `fepCtl`.
- `fepSciTimed` calls `FIOgetExpInfo` to determine the current exposure number and calls `fioWriteImPulseReg` to set the thresholder's `IPULSE_ARMNXTACQ` (write-enable) flag.
- `fepSciTimed` then enters a loop, labelled "next frame..." in Figure •, calling `FIOgetExpInfo` until the exposure numbered `fp->nextexpnum` is encountered (or exceeded). During this loop, calls are made to `FIOgetNextCmd` to handle any incoming BEP commands, and to `FIOtouchWatchdog` to prevent the Watchdog Timer from expiring while waiting for that particular frame.

- Once the desired exposure index is reached, i.e. once that frame starts to be processed by the FEP hardware, `fepSciTimed` writes a `FEPexpRec` record to the ring buffer and then calls either `FEPsciTimedEvent`, `FEPsciTimedRaw`, or `FEPsciTimedHist` to process the frame.
- `fepSciTimed` repeats the steps described in these last three bullets.
- The process continues until a `BEP_FEP_CMD_STOP` command is received from the BEP, which sets the `FP_TERMINATE` bit in *`fp->flags`*. When this occurs, `fepSciTimed` will finish processing the current frame before returning to `fepCtl`.

18.6.2 FEPsciTimedInit()

#include fepCtl.h

Scope: Science

Return type: fepCmdRetCode

Arguments

*FEPparm *fp*

Description:

This function is called from `fepSciTimed` to verify the contents of the `FEPparmBlock`, *fp->tp*, and to check whether a bias calibration has been performed. It also performs any necessary mode-dependent initialization.

If an error is detected, `FEPsciTimedInit` returns a *fepCmdRetCode* code as defined in *fepBep.h*; otherwise it returns `FEP_CMD_NOERR`.

18.6.3 FEPsciTimedEvent()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPparm *fp*

Description:

This function is called from `fepSciTimed` to process a single timed exposure frame. It initializes the following pointers to the addresses of data structures used by the hardware thresholder:

pTPlane	points to T-plane buffer
pOclk	points to overclock buffer
fp->image	points to start of first data row

and these pointers will be advanced from row to row. The `FP_EDGE_ROW` bit in *fp->flags* is set to indicate that this is the first data row, and the four `oSum[]` overclock accumulators (one per DEA output node) are zeroed.

`FEPsciTimedEvent` then loops over CCD pixel rows. Once per row, it calls `FIOgetNextCmd` to see whether the BEP is trying to command the FEP. A returned value of `TRUE` signals that a science command has been received (utility commands will be executed within `FIOgetNextCmd` and will return `FALSE`), and a call will be made to `fepHandleCmd` to process it. NOTE: only one call is made to `FIOgetNextCmd` per incoming pixel row.

The function then processes the row data. It adds the overclocks to the `oSum` accumulators. Then it reads the T-plane, 32 bits at a time, until a non-zero value is found—indicating that the hardware detected a threshold crossing. Because the 12-bit pixel and bias values must be loaded in pairs, the 32 T-plane mask bits are tested two at a time—if an even-offset bit has been set, `FEPtestEvenPixel` is called; otherwise `FEPtestOddPixel` is called.

After processing the image row, `FEPsciTimedEvent` increments the *fp->image* pointer, and recomputes the `FP_EDGE_ROW` flag. After processing the last row, the *oSum* overclock accumulators are normalized and used to derive thresholds for the next exposure, which are communicated to the hardware by calls to `FIOsetThresholdRegister` for each DEA output node.

18.6.4 FEPtestEvenPixel()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from `FEPsciTimedEvent` for every bit set in the T-plane whose bit offset is even. This pixel is located in row *irow*, column *icol*, and *fp->image* points to the first pair of pixels in that row. The corresponding pair of bias values are to be found in *fp->image*`[BIAS_OFFSET]`.

FEPtestEvenPixel first loads the 32-bit pixel and bias fields that contain the central pixel. Since the FEP is a little-endian processor, and the central pixel is stored in the lower address pair of bytes, pixel and bias will be loaded into the least-significant 16 bits of each 32-bit variable. (The most significant 16 bits of these variables will contain the pixel and bias for row *irow* and column *icol*+1.)

The pair of bias values is now validated—the bias parity flags (bits 15 and 31) are tested to determine whether a parity error has occurred in either of those locations in the bias map. If so, a call is made to `FEPsciTimedFixBias` to handle the problem. Since this may instead be a fiducial pixel, the corrected 32-bit bias pair is examined; if it still contains a BIAS_PARITY flag, the central pixel is not fiducial—the pair of values is written back to the bias map, and the central pixel is ignored. Otherwise, processing continues.

Next, a test is made to determine whether this central pixel lies on the boundary of the CCD, or if its bias value is either BIAS_BAD (indicating that it had previously been found to have a parity error) or PIXEL_BAD (indication that the pixel is a member of the bad-pixel list). In any of these situations, the pixel is ignored.

The 12-bit value of the central pixel is saved in `ev.p[1][1]` and the corresponding 12-bit bias value in *ev.b*`[1][1]`. Two threshold values are loaded from the *fp->ex.dOclk*`[]` array—*dql* and *dq*—the former referring to the 3 pixels at a lower column number than the central pixel, the latter to the remaining 6 (which may belong to a different DEA output node). To assist in later `FEPsciPixTest`

calls, the variable *val* is set equal to the difference between the center pixel value and the sum of its bias and threshold values, as discussed in Section 18.6.6.

The center pixel is now compared to those lying to its left, to its right, and to those on the preceding row, using `FEPsciPixTest`, which also saves the pixel and bias values in the appropriate elements of the *ev.p* and *ev.b* arrays. If a test fails, i.e. if the central pixel isn't a local maximum, the function returns.

Before testing the row that follows the center pixel, `FEPtestEvenPixel` calls `FIOgetExpInfo` to see whether the hardware is still processing the same frame as the software. If so, it loops over calls to `FIOgetImageMapRowPtr` until the hardware has finished processing the current row. Then the three pixels in the following row are also tested against the center pixel using `FEPsciPixTest`. This time, however, it is also necessary to check each pair of bias values for a possible parity error. If discovered, `FEPsciTimedFixBias` is called to log the occurrence, and the fixed bias value pair is stored back in the bias map.

If all 8 pixels that surround the central pixel survive the `FEPsciPixTest` criteria, the 3x3 pixel and bias arrays in the *ev* structure will have been loaded. If 5x5 mode has been selected, `FEPtestEvenPixel` calls `FEPappend5x5` to append the 16 bordering pixels and their corresponding bias values to the `FEPeventRec3x3`, transforming it into a `FEPeventRec5x5` record. Finally, `FEPtestEvenPixel` calls `FIOappendData` to write the `FEPeventRec3x3` record or `FEPeventRec5x5` record to the FEP-BEP ring buffer.

18.6.5 FEPtestOddPixel()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from `FEPsciTimedEvent` for every bit set in the T-plane whose bit offset is odd. This pixel is located in row *irow*, column *icol*, and *fp->image* points to the first pair of pixels in that row. The corresponding pair of bias values are to be found in *fp->image*`[BIAS_OFFSET]`

`FEPtestOddPixel` first loads the 32-bit pixel and bias fields that contain the central pixel. Since the FEP is a little-endian processor, and the central pixel is stored in the upper address pair of bytes, pixel and bias will be loaded into the most -significant 16 bits of each 32-bit variable. (The least significant 16 bits of these variables will contain the pixel and bias for row *irow* and column *icol*–1.)

The pair of bias values is now validated—the bias parity flags (bits 15 and 31) are tested to determine whether a parity error has occurred in either of those locations in the bias map. If so, a call is made to `FEPsciTimedFixBias` to handle the problem, the corrected 32-bit bias pair is written back to the bias map, and the central pixel is ignored. Note that pixels from odd numbered columns cannot be fiducial (*cf.* `FEPtestEvenPixel`).

Next, a test is made to determine whether this central pixel lies on the boundary of the CCD, or if its bias value is either BIAS_BAD (indicating that it had previously been found to have a parity error) or PIXEL_BAD (indication that the pixel is a member of the bad-pixel list). In any of these situations, the pixel is ignored.

The 12-bit value of the central pixel is saved in *ev.p*`[1][1]` and the corresponding 12-bit bias value in *ev.b*`[1][1]`. Two threshold values are loaded from the *fp->ex.dOclk*`[]` array—*dqr* and *dq*—the former referring to the 3 pixels at a higher column number than the central pixel, the latter to the remaining 6 (which may belong to a different DEA output node). To assist in later `FEPsciPixTest` calls, the variable `val` is set equal to the difference between the center pixel value

and the sum of its bias and threshold values, as discussed in Section 18.6.6.

The center pixel is now compared to those lying to its left, to its right, and to those on the preceding row, using `FEPsciPixTest`, which also saves the pixel and bias values in the appropriate elements of the *ev.p* and *ev.b* arrays. If a test fails, i.e. if the central pixel isn't a local maximum, the function returns.

Before testing the row that follows the center pixel, `FEPtestOddPixel` calls `FIOgetExpInfo` to see whether the hardware is still processing the same frame as the software. If it is, it loops over calls to `FIOgetImageMapRowPtr` until the hardware has finished processing the current row. Then the three pixels in the following row are also tested against the center pixel using `FEPsciPixTest`. This time, however, it is also necessary to check each pair of bias values for a possible parity error. If discovered, `FEPsciTimedFixBias` is called to log the occurrence, and the fixed bias value pair is stored back in the bias map.

If all 8 pixels that surround the central pixel survive the `FEPsciPixTest` criteria, the 3x3 pixel and bias arrays in the *ev* structure will have been loaded. If 5x5 mode has been selected, `FEPtestOddPixel` calls `FEPappend5x5` to append the 16 bordering pixels and their corresponding bias values to the `FEPeventRec3x3`, transforming it into a `FEPeventRec5x5` record. Finally, `FEPtestOddPixel` calls `FIOappendData` to write the `FEPeventRec3x3` record or `FEPeventRec5x5` record to the FEP-BEP ring buffer.

18.6.6 FEPsciPixTest

#include fepCtl.h

Scope: Science

Return type: unsigned

Arguments

```
FEPeventRec5x5 *ev

unsigned drow

unsigned dcol

unsigned val

unsigned pixel

unsigned code

unsigned bias
```

Description:

This is an `inline` function that is invoked several times within the `FEPtestEvenPixel` and `FEPtestOddPixel` functions. It masks the 12 low-order bits of `pixel` and `bias` and stores them in `p[drow][dcol]` and `b[drow][dcol]` in the ev structure. It then compares the value of (`pixel - bias`) against `val`, the corresponding value derived for the central pixel,

```
val = ev.p[1][1] - ev.b[1][1] - doclk;
```

where `doclk` represents an adjustment, quadrant by quadrant and frame by frame, for changes in average overclock. The averages from the first frame that was used in bias calibration are stored in `fp->br.bias0[]` and reported in `FEPexpRec` records, and all subsequent thresholds must be corrected by the difference between those `bias0` values and the current overclock averages. This difference is reported in the `dOclk` array in `FEPexpRec` records.

`FEPsciPixTest` evaluates to `TRUE` when the value of the pixel at (`drow, dcol`) is inconsistent with a legal event, or `FALSE` when it isn't. When the `code` argument is `TRUE` (non-zero), the comparison is "less-than-or-equal", i.e.

```
return val <= (pixel - bias);
```

This test is used for the 3 pixels whose drow index is greater than that of the central pixel, and for the pixel whose drow index is the same but whose *dcol* index is greater than the central pixel; for the remaining pixels, *code* should be set to FALSE (zero) and the comparison will be "less than", i.e.

```
return val < (pixel - bias);
```

FEPsciPixTest also returns FALSE when the bias value is either PIXEL_BAD or BIAS_BAD, i.e. when a parity error has previously been detected in the *bias* value, or when the corresponding pixel is a member of the bad pixel list.

drow and *dcol* must be in the range 0-2; the "center" pixel has *drow*=*dcol*=1; *drow*=0 refers to the row before the center pixel, *drow*=2 to the row after; *dcol*=0 to the column before, *dcol*=2 to the column after. Constants PIXEL_MASK, PIXEL_BAD, and BIAS_BAD are defined within *fepCtl.h*, and *val* and *ev* are local variables.

NOTE: In 3x3 mode, only that part of the FEPeventRec5x5 structure which is identical to a FEPeventRec3x3 will be accessed by FEPtestEvenPixel and FEPtestOddPixel.

18.6.7 FEPappend5x5()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPeventRec5x5 *ev*

*unsigned *pImage*

*FEPparm *fp*

Description:

This function is called from `FEPtestEvenPixel` and from `FEPtestOddPixel` to copy the 16 edge pixels and their corresponding bias values to the *ev->pe[]* and *ev->be[]* arrays. In entry, *pImage* points to the 32-bit word in the pixel map that contains the central pixel of the event.

Since the calling routines will not have determined whether the last of the 5 rows has yet been processed by the FEP hardware, `FEPappend5x5` first calls `FIOgetExpInfo` to see whether the hardware is still processing the same frame as the software. If so, it loops over calls to `FIOgetImageMapRowPtr` until the 5th row (*ev->row*+2) is available.

The edge pixels and their bias values are then loaded, masked, and copied to *ev->pe[]* and *ev->be[]*, taking care to detect and fix any bias parity violations via a call to `FEPsciTimedFixBias`.

18.6.8 FEPsciTimedFixBias()

#include fepCtl.h

Scope: Science

Return type: unsigned

Arguments

unsigned bias

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from FEPtestEvenPixel or FEPtestOddPixel, (or FEPappend5x5) when a parity error or fiducial pixel is encountered in either or both of the halves of a 32-bit bias word.

FEPsciTimedFixBias first determines whether fiducial pixels have been defined (*fp->nfidpix* > 0), and also whether the parity flag is set for the even-column bias value. If both are true, the *fp->fidpix* array is scanned to see if it contains the current *irow,icol* address. If so, e.g. in *fp->fidpix*[*ii*], FEPsciTimedFixBias writes to the ring buffer a FEPfidPixRec record containing the index *ii* and the pair of pixel values, and it then removes the even pixel's parity error flag from the *bias* variable.

Fiducial pixels excepted, each parity-corrupted 12-bit value in *bias* is replaced by BIAS_BAD (defined in *fepCtl.h*) and FEPsciTimedError is called (twice if both 12-bit values are bad) to reset the appropriate bits in the Bias Parity Plane and T-plane. It then calls FIOappendData to send a message of type FEP_ERROR_REC to the BEP containing *irow*, *icol*, the uncorrected *bias* value, and the current exposure number in *fp->ex.expnum*. Finally, FEPsciTimedFixBias returns the fixed-up 32-bit bias word to the caller, which has the responsibility for saving it back in the appropriate location in the bias map.

The returned value consists of the pair of input pixels and their parity error flags. If the even-column pixel was fiducial, the corresponding error flag is cleared in the returned value.

18.6.9 FEPsciTimedError()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from `FEPsciTimedFixBias` when a parity error is detected in row *irow*, column *icol* of the bias map. It resets the relevant bit in the Bias Parity Plane, turns off the corresponding bit in the T-plane, and increments the bias error count, *fp->exend.parityerrs*.

18.6.10 FEPsciTimedRaw()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPparm *fp*

Description:

This function is called from `fepSciTimed` to process a single raw CCD image frame. It initializes the following pointers to the addresses of data structures used by the hardware thresholder:

pOclk	points to start of overclock buffer
image	points to start of first data row

and these pointers will be advanced from row to row.

`FEPsciTimedRaw` then loops over CCD pixel rows. Once per row, it calls `FIOgetNextCmd` to see whether the BEP is trying to command the FEP. A returned value of `TRUE` signals that a science command has been received (utility commands will be executed within `FIOgetNextCmd` and will return `FALSE`), and a call will be made to `fepHandleCmd` to process it. NOTE: only one call is made to `FIOgetNextCmd` per incoming pixel row.

The function then processes the row data, copying the raw pixels and overclocks to a `FEPeventRecRaw` structure and then calling `FIOappendData` to copy it to the ring buffer.

18.6.11 FEPsciTimedHist()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPparm *fp*

Description:

This function is called from `fepSciTimed` to extract raw pixel histogram data from a single image frame. It initializes the following pointers to the addresses of data structures used by the hardware thresholder:

pOclk	points to start of overclock buffer
image	points to start of first data row

and these pointers will be advanced from row to row. If this is the first of a series of *fp->tp.nhist* frames, it saves the frame counter in the **fp->phist* histogram record and clears the histogram arrays.

`FEPsciTimedHist` then loops over CCD pixel rows. Once per row, it calls `FIOgetNextCmd` to see whether the BEP is trying to command the FEP. A returned value of `TRUE` signals that a science command has been received (utility commands will be executed within `FIOgetNextCmd` and will return `FALSE`), and a call will be made to `fepHandleCmd` to process it. NOTE: only one call is made to `FIOgetNextCmd` per incoming pixel row.

The routine accumulates each row of pixels into the *fp->phist->hist[nq]* histogram arrays, where *nq* represents the appropriate DEA output node index. It then examines the row's overclocks, updating the minimum (*fp->phist->omin[nq]*) and maximum (*fp->phist->omax[nq]*) values, and accumulating their sum (*osum[nq]*) and sum-of-squares. The latter are calculated in 64-bit arithmetic, using pairs of `unsigned` fields, *ossql[nq]* and *ossqh[nq]*.

After each frame, mean overclocks and variances are summed into *fp->phist->omean* and *fp->phist->ovar*, as described in Section 18.5.4.

After processing the last line of the image frame, `FEPsciTimedHist` inspects *fp->ocount[nq]* and computes the mean overclock values (*fp->phist->omean[nq]*) and their variances (*fp->phist->ovar[nq]*) for each DEA output node, calling `FIOappendData` to copy the `FEPeventRecHist` record to the ring buffer.