CSR	MASSACHUSETTS INSTITUTE OF TECHNOLOGY CENTER FOR SPACE RESEARCH CAMBRIDGE, MASSACHUSETTS 02139	
REVISION LOG	**TITLE: Software Detailed Design** **FEP Science Continuously Clocked Modes**	**DOC. NO.** **36-53225 Rev. B**

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
A	4/22/96	36-600	all	Initial released version. Incorporated comments from initial review.	RFG 5/23/96
B	6/17/96	36-623	1257, 1258	Added support to ignore "n" initial frames	6/18/96

18.0 FEP Continuously Clocked Modes (36-53225 B)

18.1 Purpose

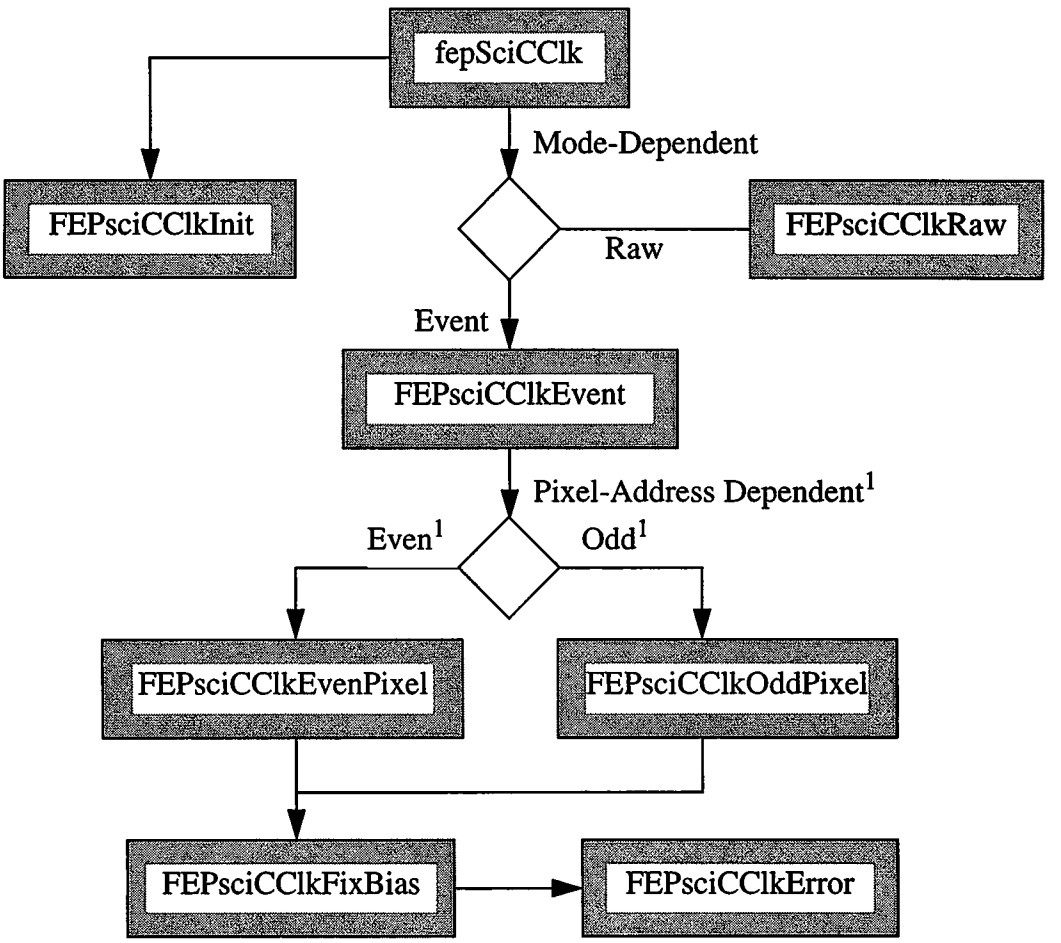
The `fepSciCClk` module, executing in the FEP, implements continuously clocked science processing, comprising either raw mode or event detection mode. It is called from `fepCtl` with a single argument, `fp`, a pointer to the `fepParm` structure. Before invoking `fepSciCClk`, the FEP must be commanded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

18.2 Uses

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

Use 1:: Identify and report 1x3 candidate X-ray events to the BEP
Use 2:: Report values of raw pixels and overclocks to the BEP

FIGURE 186. fepSciCClk 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. `fepSciCClk` makes use of the following functions which call each other in the manner shown in Figure :

- `FEPsciCClkInit`—validates the `FEPparmBlock`, *`fp->tp`*, and performs all necessary mode-dependent initialization, e.g. in event-finding mode, it checks that the bias map has been initialized.
- `FEPsciCClkEvent`—processes a single continuously clocked frame (512 rows) in event-finding mode, calling `FEPsciCClkEvenPixel` and `FEPsciCClkOddPixel` for each detected threshold crossing. The flow of control is described in the shaded region of Figure 187.
- `FEPsciCClkRaw`—processes a single continuously clocked frame (512 rows) in raw mode, reporting raw pixels and overclocks in `FEPeventRecRaw` records without any thresholding, as shown in the shaded region of Figure 187.
- `FEPsciCClkEvenPixel`—processes a threshold-crossing event possessing an even bit-offset[1] relative to the start of the T-Plane buffer. If the center pixel is a local maximum, `fepAppendRingBuf` is called to copy a `FEPeventRec1x3` record to the ring buffer.
- `FEPsciCClkOddPixel` —processes a threshold-crossing event possessing an odd bit-offset[1] relative to the start of the T-Plane buffer.If the center pixel is a local maximum, `fepAppendRingBuf` is called to copy a `FEPeventRec1x3` record to the ring buffer.
- `FEPsciCClkPixTest`—is an `inline` function (and therefore not shown in Figure) that is called several times within `FEPsciCClkEvenPixel` and `FEPsciCClkOddPixel` to determine whether the central pixel is less than (or equal to) one of its neighbors.
- `FEPsciCClkFixBias`—is called if a parity error is discovered in one or both of a pair of bias map values. It locates an undamaged bias value from the other values in the same column of the bias map, calls `FEPsciCClkError` to reset the parity plane and T-plane bits, calls `fepAppendRingBuf` 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.

1. The distinction made between even and odd pixels is purely for programming efficiency. As a consequence of the FEP hardware architecture, it must access pixel and bias values in pairs.

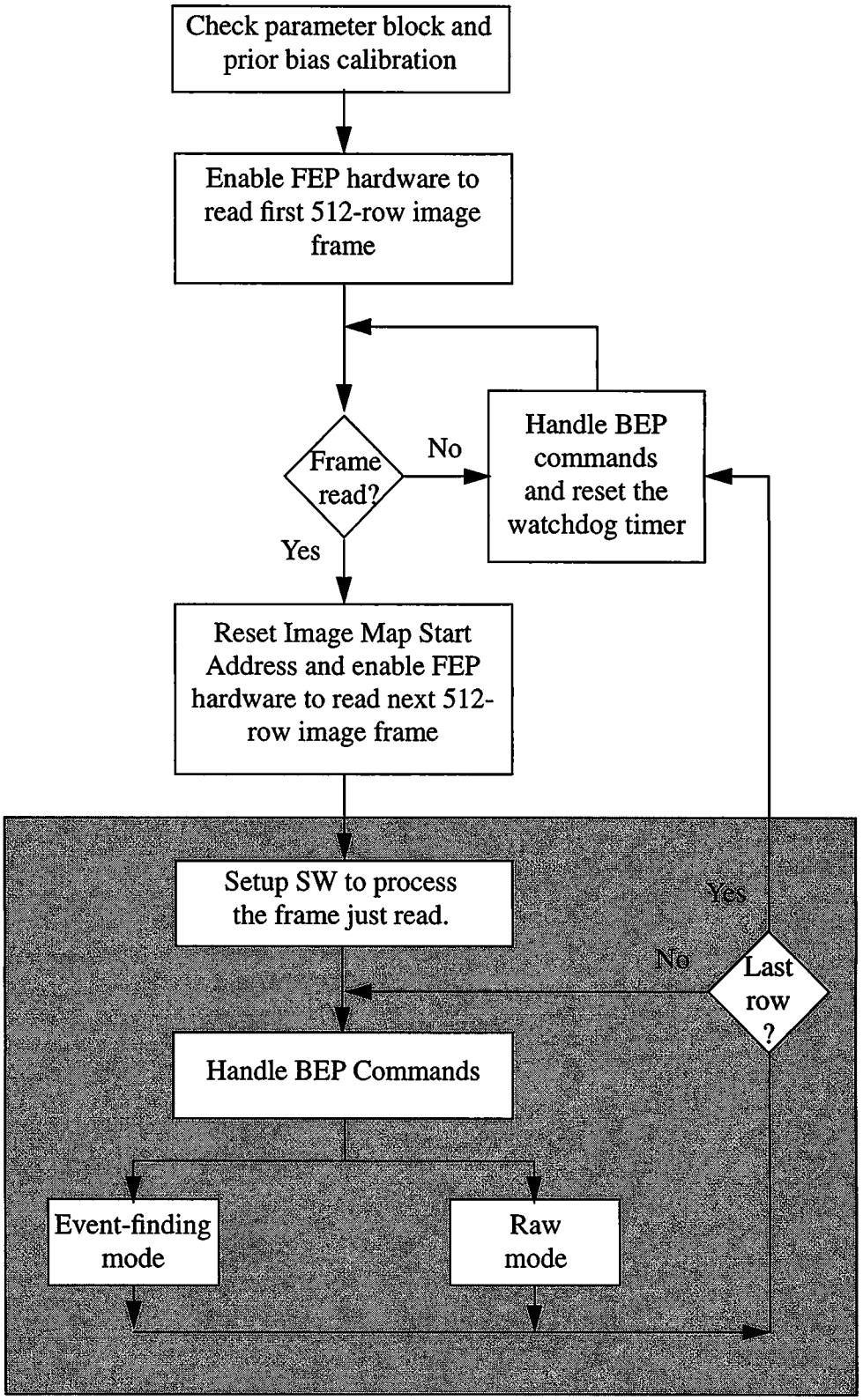


FIGURE 187. The flowchart of fepSciCClk

18.4 Global Variables

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

bepCmd	latest command received from BEP	
br	bias calibration parameters	
bias0[4]	average nodal overclocks for first bias frame	
biassum	sum of the 4 *bias0* values	
colshft	bit shift to transform column index to node index	
ex	current FEPexpRec record	
bias0[4]	copy of *fp->br.bias0* array	
dOclk[4]	overclock changes for each output node	
expnum	current "exposure" (i.e. VSYNC frame pulse) number	
timestamp	microsecond timer value at start of *expnum*	
exend	current FEPexpEndRec record	
expnum	current "exposure" (i.e. VSYNC frame pulse) number	
parityerrs	number of corrected parity errors in the frame	
thresholds	number of threshold crossings in the frame	
flags	flag bits used:	
	FP_SUSPEND	BEP has sent BEP_FEP_CMD_SUSPEND
	FP_TERMINATE	BEP has sent BEP_FEP_CMD_STOP
image	pointer to start of current 512-row frame	
lastcol	index of last frame column	
nextexpnum	the next frame index that the FEP is to process	
quadrants	the number of DEA output nodes being sampled	
tp	frame 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.	

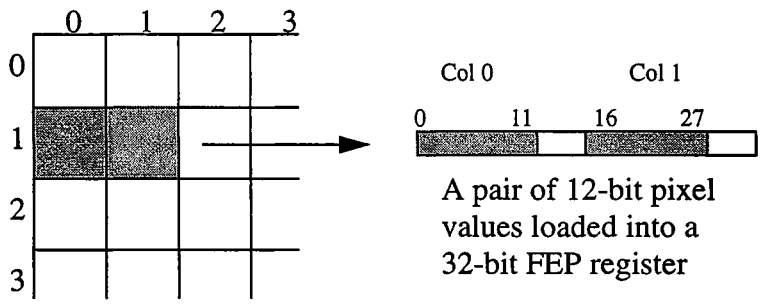
18.5 Scenarios

The following paragraphs describe the basic functions performed in continuously clocked mode. All Continuous Clocking modes wait until an image frame (a block of DEA output pixels delimited by VSYNC flags, typically comprising 512 image rows) is processed and copied to the ring buffer before commanding the hardware thresholder to read the next frame, thereby ensuring that no partial frames are generated.

18.5.1 Use 1: Report 1x3 Events

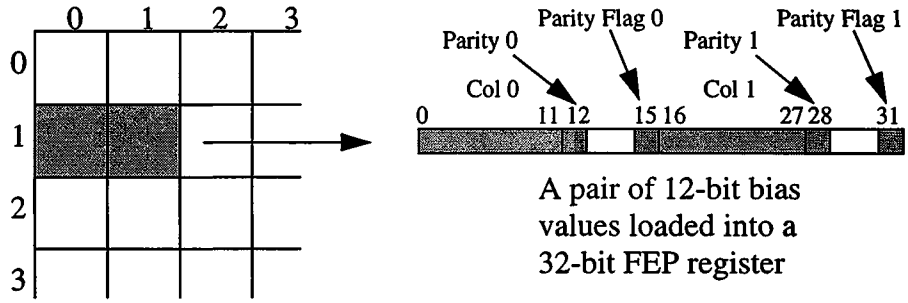
- This mode is characterized by *fp->tp.type* == FEP_CCLK_PARM_1x3. After calling FEPsciCClkInit to check that the bias array contains appropriate values and to load the hardware registers, FEPsciCClk calls fepEnableNextFrame and waits for 512-row frames to arrive. The first *fp->tp.initskip* frames are ignored.
- Once a valid frame arrives, fepSciCClk calls FIOsetImageMapRowStart and fepEnableNextFrame to enable the hardware to write the next 512 rows into the other half of the image map. fepSciCClk then copies the exposure number and arrival time of the previous frame into a FEPexpRec structure and calls fepAppendRingBuf to copy it to the ring buffer. It then calls FEPsciCClkEvent to process the frame.
- FEPsciCClkEvent examines input rows one at a time, calling FIOgetNextCmd to handle any incoming BEP commands. Since the hardware is writing into the *other* 512 rows of image map, it is not necessary to call FIOgetImageMapRowIndex while processing the frame (*cf.* fepSciTimedEvent in Section 18.6.3).
- FEPsciCClkEvent 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 FEPsciCClkEvenPixel or FEPsciCClkOddPixel 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 188. The Relation Between Image Pixels and FEP Register Values



- The two pixel-testing routines examine the pixels on either side of the pixel that triggered the hardware thresholder, as illustrated in Figure 188. Pixels in the bad column list (with a bias value of 4095) are ignored. Otherwise, if the relative value (pixel-minus-bias) of the center pixel exceeds that of the pixels on either side, or equals that of the pixel on its left, `fepAppendRingBuf` is called to report the 3 pixel- and bias-values to the ring buffer in a `FEPeventRec1x3` record.

FIGURE 189. The Relation Between the Bias Map and FEP Register Values



- Whenever either `FEPsciCClkEvenPixel` or `FEPsciCClkOddPixel` load a pair of bias values, as illustrated in Figure 189, they inspect the 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, `FEPsciCClkFixBias` is called to (a) locate an undamaged copy of the bias value from the other values in that column of the bias map, (b) reset the bias parity bit and the T-plane bit, and (c) return the corrected bias value. The caller (`FEPsciCClkEvenPixel` or `FEPsciCClkOddPixel`) then stores the corrected value back into the bias map and continues to process the event.
- If all bias values within a given column are found to contain parity errors, the event will be rejected.
- After processing the last row of pixels, `FEPsciCClkEvent` normalizes the *oSum*`[]` values and calls `FIOsetThresholdRegister` to update the hardware threshold registers.
- Finally, `fepAppendRingBuf` is called to copy a `FEPexpEndRec` (end-of-frame) record to the ring buffer.

18.5.2 Use 2: Report Raw Pixels

- This mode is characterized by *`fp->tp.type`* == `FEP_CCLK_PARM_RAW`. It begins in an identical manner to the event-finding modes described in the previous sections, except that `fepSciCClk` calls `FEPsciCClkRaw` to process each 512-row frame.
- `FEPsciCClkRaw` 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.

18.6 Specification

This section describes the functions that are local to the `fepSciCClk` unit. The only external is `fepSciCClk` 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 fepSciCClk()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPparm *fp*

Description:

`fepSciCClk` 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. In particular, it controls the double-buffering of input data—the image map is split logically into two 512-row buffers and the FEP hardware is told to write into one while the software is processing the other. `fepSciCClk` performs the following actions:

i) Calls `FEPsciCClkInit` 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 `FEPsciCClkInit` returns `FEP_CMD_NOERR`, `fepSciTimed` calls `fepAckCmd` and continues. Otherwise, it calls `fepNackCmd` to pass the `fepCmdRetCode` error code to the BEP, indicating that the command has failed, and `fepSciCClk` then returns to `fepCtl`. Error codes are defined in *fepBep.h* (see Section 4.10).

ii) Calls `fepEnableNextFrame` to enable the hardware to read the first 512 rows of image pixels into the image buffer.

iii) Loops over frames of 512 rows of image pixels.

iv) Enters a loop (see the upper half of Figure 187), 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. *fp->flags* is inspected: if the `FP_TERMINATE` bit is set, `fepSciCClk`

returns immediately—the science run is over; if the `FP_SUSPEND` bit is set, `fepSciCClk` loops until the BEP sends it a `BEP_FEP_CMD_RESUME` command.

v) Calls `FIOsetImageMapRowStart` and `fepEnableNextFrame` to enable the hardware to read the next 512 rows into the *other* half of the image buffer.

vi) Calls either `FEPsciCClkEvent` or `FEPsciCClkRaw` (according to the value of *`fp->tp.type`*) to process the 512 rows that were read by the hardware *prior* to step v).

vii) Branches back to step iii), above. The loop over input frames continues until `FIOgetNextCmd` receives a `BEP_FEP_CMD_STOP` command from the BEP, which sets the `FP_TERMINATE` bit in *`fp->flags`*. `fepSciCClk` will finish processing the current frame before returning to `fepCtl`.

18.6.2 FEPsciCClkInit()

#include fepCtl.h

Scope: Science

Return type: fepCmdRetCode

Arguments

*FEPparm *fp*

Description:

This function is called from `fepSciCClk` 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. When in event-detection mode (*fp->tp.type* == `FEP_CCLK_PARM_1x3`), hardware thresholding and bias parity error detection are enabled. Otherwise, both are disabled. Overclock processing is always enabled.

The following *fp->tp* and *fp->br* fields are checked:

Tests Applied	*Error code returned if the test fails*
quadcode ∈ `fepQuadCode`	`FEP_CMD_ERR_QUAD_CODE`
0 < *nrows* ≤ `CCLK_NROWS`	`FEP_CMD_ERR_NROWS`
0 < *ncols*, and *ncols* even	`FEP_CMD_ERR_NCOLS`
quadrants * *ncols* ≤ `MAX_NCOLS`	`FEP_CMD_ERR_NCOLS`
noclk ≤ `MAX_NOCLK`	`FEP_CMD_ERR_NOCLK`
$\sum_{i=0}^{3} br.bias0[i] \equiv br.biassum$	`FEP_CMD_ERR_NO_BIAS`

If an error is detected, `FEPsciCClkInit` returns the appropriate *fepCmdRetCode* value, as defined in *fepBep.h* (see Section 4.10); otherwise it returns `FEP_CMD_NOERR`.

18.6.3 FEPsciCClkEvent()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPparm *fp*

Description:

This function is called from `fepSciCClk` to process a single continuously clocked frame. It initializes the following row pointers to the addresses of data structures used by the hardware thresholder:

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

and these pointers will be advanced from row to row. Note that, in step with the double buffering, *pTPlane* and *fp->image* will alternately point to the start and to the middle of their respective buffers. The four `oSum[]` overclock accumulators (one per DEA output node) are zeroed.

`FEPsciCClkEvent` 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 input row.

The function then processes the row data. It adds the overclocks to the `oSum` accumulators, and then examines the T-plane, 32 bits at a time, until a non-zero value is found—indicating that the hardware detected a threshold crossing or bias map parity error. 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, `FEPsciCClkEvenPixel` is called; otherwise `FEPsciCClkOddPixel` is called.

After processing the image row, `FEPsciCClkEvent` increments the row pointers. After the last row of the frame, 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 FEPsciCClkEvenPixel()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from `FEPsciCClkEvent` for every bit set in the T-plane whose bit offset is even. The associated 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]`.

`FEPsciCClkEvenPixel` 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 `FEPsciCClkFixBias` to handle the problem, and the corrected 32-bit bias pair is written back to the bias map. Unlike the situation in timed exposure mode, when the software is merely able to report the damaged pixel, in continuous clocking mode the value can actually be repaired since each row of the bias map should be identical. Therefore, once the bias value has been updated, `FEPsciCClkEvenPixel` continues execution with the corrected value.

Next, a test is made to determine whether this central pixel lies on the left- or right-hand edge of the CCD, or if its bias value is PIXEL_BAD (indicating 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]` of a local `FEPeventRec1x3` structure, and the corresponding 12-bit bias value in *ev.b*`[1]`. To assist in later `FEPsciCClkPixTest` calls, the variable *val* is set equal to the difference between the center pixel value and its bias values, as dis-

cussed in Section 18.6.6.

The center pixel is now compared to those lying to its left and right, using `FEPsciCClkPixTest`, 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. When testing the pixel lying to the left, *`val`* must be adjusted for any change in average overclock if the two pixels were generated by different CCD output nodes. This change is simply the difference between the corresponding node values in the *`fp->ex.dOclk[]`* array.

If the pixels on either side of the central pixel survive the `FEPsciCClkPixTest` criteria, the 1x3 pixel and bias arrays in the *`ev`* structure will have been loaded. `FEPtestEvenPixel` then calls `fepAppendRingBuf` to write the `FEPeventRec1x3` record to the FEP-BEP ring buffer.

18.6.5 FEPsciCClkOddPixel()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from FEPsciCClkEvent for every bit set in the T-plane whose bit offset is odd. The associated 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].

FEPsciCClkOddPixel 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 FEPsciCClkFixBias to handle the problem, the corrected 32-bit bias pair is written back to the bias map. Unlike the situation in timed exposure mode, when the software is merely able to report the damaged pixel, in continuous clocking mode the value can actually be repaired since each row of the bias map should be identical. Therefore, once the bias value has been updated, FEPsciCClkOddPixel continues execution with the corrected value.

Next, a test is made to determine whether this central pixel lies on the left- or right-hand edge of the CCD, or if its bias value is PIXEL_BAD (indicating 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] of a local FEPeventRec1x3 structure, and the corresponding 12-bit bias value in *ev.b*[1]. To assist in later FEPsciCClkPixTest calls, the variable *val* is set equal to the difference between the center pixel value and its bias value, as dis-

cussed in Section 18.6.6.

The center pixel is now compared to those lying to its left and right, using `FEPsciCClkPixTest`, 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. When testing the pixel lying to the right, *`val`* must be adjusted for any change in average overclock if the two pixels were generated by different CCD output nodes. This change is simply the difference between the corresponding node values in the *`fp->ex.dOclk`*`[]` array.

If the pixels on either side of the central pixel survive the `FEPsciCClkPixTest` criteria, the 1x3 pixel and bias arrays in the *`ev`* structure will have been loaded. `FEPsciCClkOddPixel` then calls `fepAppendRingBuf` to write the `FEPeventRec1x3` record to the FEP-BEP ring buffer.

18.6.6 FEPsciCClkPixTest

#include fepCtl.h

Scope: Science

Return type: unsigned

Arguments

*FEPeventRec1x3 *ev*

unsigned dcol

int val

unsigned pixel

unsigned mode

unsigned bias

Description:

This is an `inline` function that is used several times within the `FEPsciCClkEvenPixel` and `FEPsciCClkOddPixel` functions. It masks the 12 low-order bits of `pixel` and `bias` and stores them in `p[`*dcol*`]` and `b[`*dcol*`]` in the `ev` structure. It then compares the value of (*pixel - bias*) against `val`.

`FEPsciCClkPixTest` evaluates to `TRUE` when the value of the pixel at *dcol* is inconsistent with a legal event, or `FALSE` when it isn't. Illegal events are those for which the corresponding bias value is `PIXEL_BAD`, i.e . when the corresponding pixel is a member of the bad column list, or those that pass the test defined by the *mode* parameter, which is either `LE` ("less-than-or-equal"), in which an illegal pixel satisfies

val <= (pixel - bias)

or LT ("less than"), when an illegal pixel satisfies

val < (pixel - bias)

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

18.6.7 FEPsciCClkFixBias()

#include fepCtl.h

Scope: Science

Return type: unsigned

Arguments

unsigned bias

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from `FEPsciCClkEvenPixel` or `FEPsciCClkOddPixel`, when a parity error is detected in either or both of the halves of a 32-bit `bias` word. Each corrupted 12-bit value is replaced by the first non-corrupted value in the same column *icol* of the bias map, 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 `fepAppendRingBuf` 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, `FEPsciCClkFixBias` 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. NOTE: since `bias` represents a pair of values, *icol* must necessarily be even.

18.6.8 FEPsciCClkError()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

unsigned val

unsigned irow

unsigned icol

*FEPparm *fp*

Description:

This function is called from `FEPsciCClkFixBias` when a parity error is detected in row *irow*, column *icol* of the bias map. It computes the parity of the 12-bit bias value, *val*, and resets the relevant bit in the Bias Parity Plane, turns off the corresponding bit in the T-plane, and increments the bias error counter, *fp->exend.parityerrs*.

18.6.9 FEPsciCClkRaw()

#include fepCtl.h

Scope: Science

Return type: void

Arguments

*FEPparm *fp*

Description:

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

pOclk	points to start of overclock buffer
fp->image	points to start of first data row in image buffer

and these pointers will be advanced from row to row. Note that, in step with the double buffering, *fp->image* will alternately point to the start and to the middle of the image buffer.

`FEPsciCClkRaw` 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 `fepAppendRingBuf` to copy it to the ring buffer.