		ENGINEERING CHANGE ORDER		<u>ECO No.</u> <u>36-1045</u>	
CENTER FOR SPACE RESEARCH MASSACHUSETTS INSTITUTE OF TECHNOLOGY					
DWG. NO.		NEW REV.		DRAWING TITLE	
36-58010		F		Flight Software Standard Patch Release E, Optional Release F	
REASON FOR CHANGE: Add optional patch txings to accumulate threshold crossings and set special bi-level value when conditions indicate an unacceptable level of background radiation.					
DESCRIPTION OF CHANGE: The txings patch replaces <code>EventExposure::copyExpEnd()</code> to accumulate threshold crossings from all FEP frames that were processed by the pixel thresholders, and replaces <code>Leds::show()</code> to test the integrated crossing rates and, if these exceed pre-set thresholds, set the ACIS bi-levels to a special value, <code>LED_BOOT_SPARE1</code> , that can be recognized by the OBC as a command to "safe" the science instruments.					
	SIGNATURE	DATE	REMARKS:		
ORIGINATOR	RFG	03/02/11	Reviewed and signed-off		
MECHANICAL					
ELECTRICAL					
SOFTWARE					
STRUCTURE					
FABRICATION					
SCIENCE					
SYSTEMS ENG.					
QUALITY					
PROJ. ENGINEER					
DEPUTY PM					
PROJ. MANAGER					

Existing ACIS Flight Software Patches

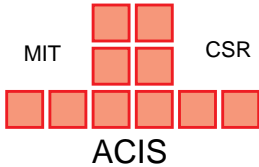
ID	Name	Rev	Size	Part	ECO	SPR
Standard Release E						
i	corruptblock	A	16	36-58030.01	994	113
ii	digestbiaserror	A	64	36-58030.02	995	116
iii	histogramvar	A	16	36-58030.03	999	115
iv	biastiming	A	112	36-58030.04	993	117
v	rquad	A	16	36-58030.14	1000	121
vi	histogrammean	A	156	36-58030.15	996	123
vii	zaplexpo	A	64	36-58030.16	997	122
viii	condock	A	640	36-58030.17	1012	127
ix	fepbiasparity2	A	504	36-58030.19	1015	130
x	cornermean	A	32	36-58030.21	1017	128
xi	tlmbusy	A	344	36-58030.29	1033	138
xii	buscrash	A	296	36-58030.30	1034	140
xiii	badpix	A	60	36-58030.31	1037	141
xiv	buscrash2	B	428	36-58030.32	1041	142
Optional Release F						
1	smtimedlookup	A	3712	36-58030.24	1025	N/A
2	eventhist	B	5908	36-58030.05	1025	N/A
3	cc3x3	B	4636	36-58030.06	1018	120,124,126
4	ctireport1	A	5452	36-58030.25	1026	N/A
5	ctireport2	A	2784	36-58030.26	1026	N/A
6	compressall	A	2368	36-58030.27	1027	134
7	untricklebias	B	1740	36-58030.28	1028	133
8	reportgrade1	A	816	36-58030.22	1021	131,132
9	txings	A	3128	36-58030.33	1044	N/A
leaf	teignore	A	36	36-58030.09	1003	N/A
leaf	ccignore	A	36	36-58030.10	1004	N/A
Under Development						
12	hybrid	03	6104	36-58030.13	1010	N/A
13	fepbiasparity1	02		36-58030.18	1014	N/A
14	squeegy	06	4412	36-58030.23	1023	N/A
15	forcebiastrickle	01	N/A	36-58030.29	1024	133
Engineering Unit Utility Patches						
10	tlmio	02	10312	36-58030.07	1010	N/A
11	printswouse	01	7224	36-58030.08	986	N/A
leaf	deaeng	02	2604	36-58030.11	1010	N/A
leaf	dearepl	02	556	36-58030.12	1010	N/A

Status of Patch Release E, Optional Revision F

Name	Part Number	Description	Typos ^a	RIDs ^b	Status
<i>txings</i>	36-58030.33 (ECO 36-1044)	Report high background radiation levels.	0	0	Reviewed and signed off
S/W Review	36-58020 (ECO 36-1045)	Documentation accompanying the individual patch ECOs	20	0	Reviewed and signed off
Certification	36-58021.04 (ECO 36-1046)	Documentation describing the multi-patch certification tests	1	0	Reviewed and signed off

a. typographical errors in the documentation

b. review item discrepancies—requiring changes to the patch code and/or test procedures

	ENGINEERING CHANGE ORDER	<u>ECO No.</u> <u>36-1044</u>
---	---------------------------------	--

CENTER FOR SPACE RESEARCH
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

DWG. NO.	NEW REV.	DRAWING TITLE
36-58030.33	A	Flight S/W patch to report high background radiation levels

REASON FOR CHANGE:

Gradual deterioration of the EPHIN particle detector will likely leave ACIS exposed to damage from low-energy charged particles. A study of 10 years of ACIS data and more than 50 radiation alerts shows that many periods of high background are associated with increasing ACIS threshold crossing rates, which can be monitored by ACIS flight software with relative ease.

DESCRIPTION OF CHANGE:

The *txings* patch accumulates threshold crossing rates for all active CCDs of each type (front-and back-illuminated) during any timed exposure or continuous clocking run that uses the FEP pixel thresholders. When the averaged rates are found to be increasing for a specified number of integration periods, the software bi-levels are set to LED_BOOT_SPARE1, an otherwise unused value, so that the OBC can respond by "safeing" the science instruments.

	SIGNATURE	DATE	REMARKS:
ORIGINATOR	RFG	03/02/11	Reviewed and signed-off
MECHANICAL			
ELECTRICAL			
SOFTWARE			
STRUCTURE			
FABRICATION			
SCIENCE			
SYSTEMS ENG.			
QUALITY			
PROJ. ENGINEER			
DEPUTY PM			
PROJ. MANAGER			

To: ACIS Science Operations Team
From: Peter Ford, NE80-6071 <pgf@space.mit.edu>
Date: April 15th 2011
Subject: Using ACIS to detect and report high radiation conditions (v 1.3)

1. Introduction

With the continuing degradation of the EPHIN detector on board the Chandra X-Ray Observatory, we are evaluating alternatives to the major function of that instrument: to report the electron and proton flux over a range of energies, permitting the observatory to take the actions necessary to preserve its instruments during times of high solar activity.

A recent analysis [Grant *et al.*, 2010] has shown that, in certain circumstances, the signature of solar events can be detected within the counts of CCD threshold crossings that are included in downlinked telemetry. While that work continues, the current report examines whether it would be practical to develop a patch to the existing ACIS flight software that could monitor threshold crossings and communicate an alarm to the Chandra On-Board Computer (OBC).

2. ACIS Threshold Counts

Each active ACIS CCD sends its digitized pixel values to a Front End Processor (FEP), whose firmware discriminates between values that are above and below a threshold. This threshold is determined by the sum of (a) the value corresponding to that pixel in a pre-computed bias map; (b) a constant *eventThreshold*¹ that is uplinked to ACIS within the parameter block that controls the science run; and (c) a correction factor computed from difference between the average “overclock” values from the previous CCD frame and those from the first CCD frame that was used to compute the bias map. The three factors therefore represent the value expected for an “eventless” pixel, plus an estimate of the variance in that value, plus a correction for the drift in the DC sampling level during the course of the run.

The thresholding firmware fills a memory buffer composed of 32-bit words, each bit of which maps to 32 input pixels and is given the value 0 or 1 according to whether that pixel’s value is less than the threshold or not. The FEP’s software need only read one word from this buffer to determine whether any of the corresponding 32 pixels are “interesting”, thereby greatly speeding up its work of locating event candidates, *i.e.*, local pixel maxima. During this process, the software retains a count of the number of threshold crossings and sends this to the BEP after processing each CCD frame.

The only filtering that is performed on the threshold crossings is in the choice of *eventThreshold* in the run’s parameter block. Each output node of each CCD can be assigned a different value, but in fact these have not changed since launch—20 ADU for back-illuminated CCDs (BI: S1 and S3), and 38 for front-illuminated (FI: I0–I3, S0, S2, S4, S5)—except when observing optically-bright sources, *e.g.*, Jupiter and Saturn, when *eventThreshold* for the observing CCD (typically S3) is increased to prevent the optical signal from contributing false triggers. The choice of bias algorithm also systematically affects the threshold count. For timed-exposure runs, the algorithm has not changed since launch: some minor adjustments have been made to its parameters, but these have had no observed effect on the threshold rate. For continuous-clocking runs, the bias algorithm was changed for FI CCDs in 2005, but again the average threshold rate was unaffected.

The metric that best represents the threshold crossing rate is the number per frame, divided by the number of pixels exposed, and divided by the exposure time. The simplest algorithm therefore keeps a running sum of threshold crossings and a second running sum of exposure times. At fixed intervals, the first is divided by the

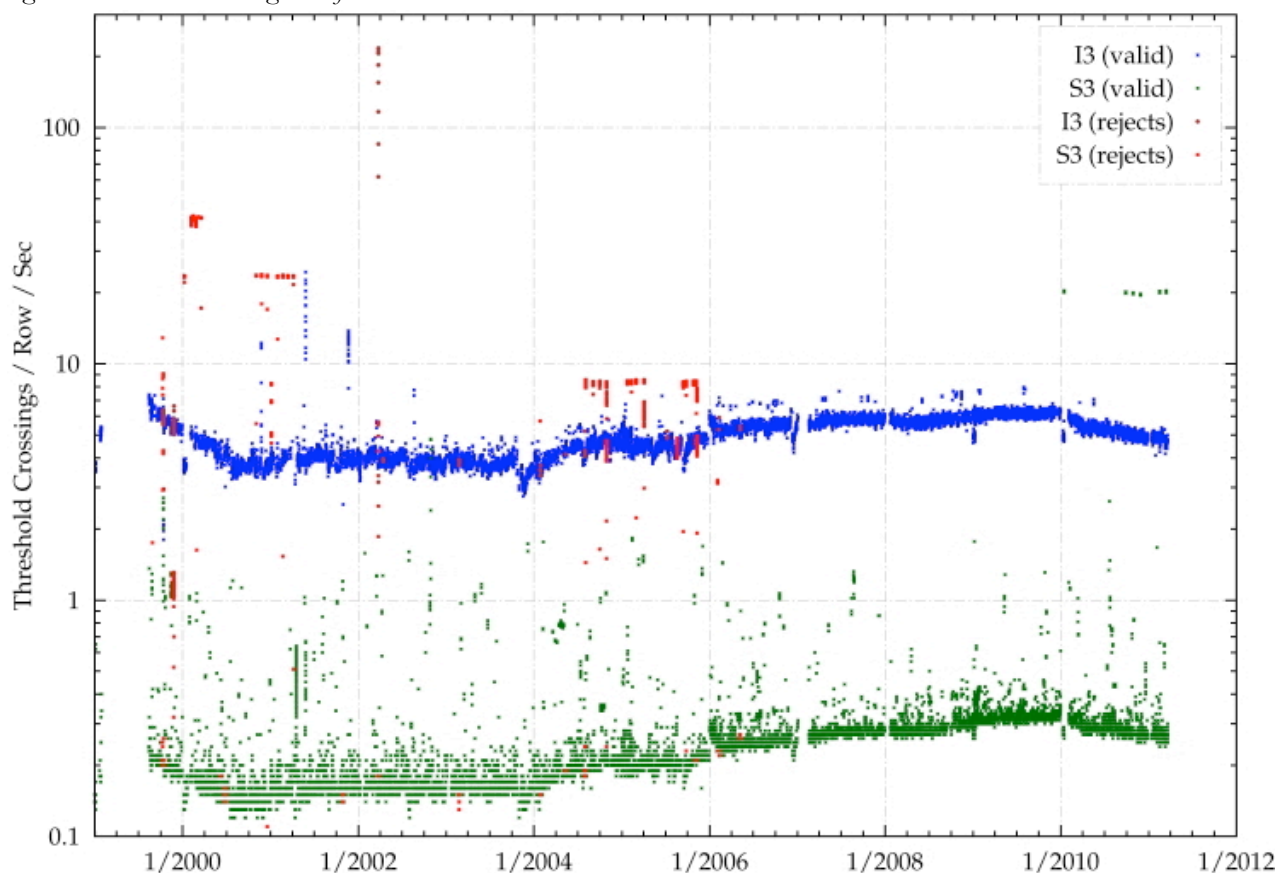
¹ In this report, names of command and telemetry packets and their fields are written in *italics*; the names of classes, methods, objects, and variables of the ACIS flight software are written in a **typewriter font**.

second, yielding the average number of crossings per second per pixel row, and when the rising threshold count rates exceed a predetermined value, an alarm is posted. In (un-summed) continuous clocking mode, there are always 524288 pixels per exposure “frame” (512 rows of 1024 pixels), and they are always exposed for 2.9184 seconds, as determined by the ACIS clock, which is accurate to a part in 10^5 . In timed-exposure mode, an exposure frame always consists of 1024 pixels per row, but the number of rows is specified by one plus the value of the *subarrayRowCount* field in the parameter block. Similarly, the exposure time of the first frame is $0.1 * \text{primaryExposure}$ seconds, followed by *dutyCycle* frames at $0.1 * \text{secondaryExposure}$ seconds, and so on. To each of these exposure times, it is necessary to add 0.04104 seconds to account for the time taken to transfer the pixels from the image store to the frame store.

3. Observed High Background Events

To choose suitable parameters for a radiation filter, all science runs in which ACIS ran in event-mode in the focal plane were examined and their threshold crossing rates were averaged over 300 second intervals. The rates for CCD_I3 (front-illuminated) and CCD_S3 (back-illuminated) are shown in Figure 1. To reduce the number of points plotted, all I3 rates that exceed 10 crossings/row/sec are shown, as are all S3 rates above 1.0 crossings/row/sec. In addition, the rates for every 50th interval, irrespective of rate, are also plotted in order to show the normal range of crossing rates for these CCDs. The points are colored to distinguish those from “valid” and “rejected” runs. The latter include observations of the Crab Nebula and Jupiter (usually with S3) and two periods of FEP hardware anomalies (T-plane latch-up and bias parity plane anomalies.)

Figure 1: Threshold crossing rates for I3 and S3



The gentle rise in the rates for “valid” runs from 2001 through 2009 has also been seen in background event rates, and is probably due to the increase in cosmic ray background during that part of the solar cycle. The high “rejected” I3 rates in 1999 and in early 2002 are due to the FEP hardware anomalies.

4. A Possible Trigger Algorithm

The FEP hardware anomalies can be filtered out by ignoring any exposure in which the number of threshold crossings exceeds some fraction of the available pixels. A simple radiation threshold algorithm (ignoring summed-pixel modes) takes the following form:

1. For each exposure of each active CCD:
 - a. Ignore if $thresholdPixels > MAX_TX_PER_ROW * (subarrayRowCount+1)$ (timed exposures)
Ignore if $thresholdPixels > MAX_TX_PER_ROW * 512$ (continuous clocking)
 - b. Add $thresholdPixels$ to crossings accumulator for this CCD, $threshold[nccd]$.
 - c. Add exposure time to the time accumulator for this CCD, $exposure[nccd]$, i.e.,
 $0.1 * primaryExposure + 0.04104$ secs (timed exposures)
 or $0.1 * secondaryExposure + 0.04104$ secs if $(exposureNumber \% (dutyCycle+1))$
 or 2.9184 seconds (continuous clocking)
2. After a suitable integration time, MINUTES minutes:
 - a. Compute the average threshold rate (r) in crossings per row per second, i.e.,
 $r = threshold[nccd] / exposure[nccd] / (subarrayRowCount + 1)$ (timed exposures)
 $threshold[nccd] / exposure[nccd] / 512$ (continuous clocking)
 - b. Inspect the average rates of all active front-illuminated CCDs.
If they all exceed a preset threshold, $RATE_LIMIT[0]$, save their average; otherwise save 0.
 - c. Inspect the average rates of all active back-illuminated CCDs.
If they all exceed a preset threshold, $RATE_LIMIT[1]$, save their average; otherwise save 0.
3. If either the front-illuminated or the back-illuminated average rate is non-zero, and has increased by more than $TX_INCR[i]$ ($i=0,1$) for each of $TRIGGER_COUNT$ consecutive integration intervals, sound the alarm.

The following table shows the triggers that resulted from running the algorithm over all OBSIDs, when ACIS was in the focal plane, from November 1999 through April 2011, for the optimum choice of parameters: $MAX_TX_PER_ROW=512$, $MINUTES=5$, $RATE_LIMIT[0]=6.75+0.025*y$, $RATE_LIMIT[1]=0.4+0.02*y$, $TX_INCR[0]=TX_INCR[1]=0.02$, and $TRIGGER_COUNT=5$. y is the number of years past 2000.0. The FB column indicates the type and number of CCDs that caused the trigger, and the N column shows how many consecutive 5-minute intervals were found to exceed the threshold criteria.

	Date	Phase	Run	OBSID	Mode	FB	N	Target
1.	2000-11-24	acis6	41	2344	Te3x3	FI5	9	HDF_NORTH
2.	2001-04-03	acis7	180	1578	Te3x3	FI4	19	NGC4111 (RADMON)
3.	2001-09-24	acis10	79	1890	Te3x3	FI4	5	HD_93497 (RADMON)
4.	2001-11-04	acis10	165	2010	Te3x3	BI2	7	PSR0628-28 (TPLANE_LATCHUP) (RADMON)
5.	2001-11-21	acis11	29	3389	Te5x5	FI4	5	HDF-N (RADMON)
6.	2002-03-18	acis12	175	3463	Cc3x3	BI2	5	RX_J170930.2-26 (SCS107)
7.	2002-04-21	acis13	40	61227	Te3x3	FI5	5	FAINT_MODE_I (RADMON)
8.	2002-08-21	acis14	210	4365	Te3x3	FI5	5	GROTH-WESTPHAL_
9.	2002-08-23	acis14	217	2783	Te5x5	BI1	8	SNR_N157B (RADMON)
10.	2004-11-07	acis27	10	6152	Te5x5	BI2	5	M101 (SCS107)
11.	2005-09-07	acis31	107	5760	Te5x5	FI5	7	CL0216-1747 (RADMON)
12.	2006-12-13	acis38	163	58650	Te3x3	BI2	5	CTI_CAL_S
13.	2009-08-28	acis56	38	56867	Te3x3	FI5	6	CTI_CAL_I

The filter triggered during 13 runs, 7 of which were terminated by EPHIN and 2 by ground command due to high radiation fluxes. EPHIN E1300 rates during the remaining OBSIDs, 2344, 4365, 58650, and 56867 were only slightly below the RADMON trigger threshold.

5. Sending a Trigger to the OBC

All communication to and from the ACIS instrument passes through a Remote Command & Telemetry Unit (RCTU). Most output channels are assigned to thermistors and to sensors on the Power Supply and Mechanism Controller (PSMC), and are inaccessible to the flight software, whose principal output flows through a serial digital interface in a structured series of telemetry packets. While it would be simple to reprogram a currently unused field in one of the packet formats to use as a radiation alarm, the packets are written asynchronously to the RCTU, making it next to impossible for the OBC to locate a particular packet field within the 24 kbps serial channel.

Table 1. ACIS Bi-Level Bit Assignments

Bit Symbol	Bit Pattern	Instrument State
1STAT7ST	1 0 0 0 0 0 0 0	BEP input FIFO: 0=empty, 1=not empty
1STAT6ST	0 1 0 0 0 0 0 0	BEP FIFO: 0=full, 1=not full
1STAT5ST	0 0 1 0 0 0 0 0	BEP CPU state: 0=halted, 1=running
1STAT4ST	0 0 0 1 0 0 0 0	ID of active BEP: 0=A, 1=B
1STAT3ST	0 0 0 0 1 0 0 0	Software state: 0=running, 1=loading
1STAT2ST	0 0 0 0 0 1 0 0	Software startup: 0=watchdog, 1=normal
1STAT1ST	0 0 0 0 0 0 1 0	Science run: 0=running, 1=idle
1STAT0ST	0 0 0 0 0 0 0 1	Software toggle: alternating 0 or 1

Happily, there is an alternative in the form of a set of 8 bi-levels, 1-bit signal channels that are sent to the RCTU through a separate path. These can easily be accessed by the OBC in much the same way that it monitors the EPHIN output channels. The 8 bi-levels are shown in Table 1. Four (1STAT7ST–1STAT4ST) are controlled by DPA hardware and report on the status of the DPA and its input command FIFO. The remainder (1STAT3ST–1STAT0ST) are controlled by the flight software executing within the BEP.

The 16 combinations that can be assigned to the 4 bits under flight software control are shown in Table 2. Note that the BEP only uses the first 8 values during normal operations and 6 of the remainder when booting up, leaving two values completely unused. Chandra telemetry from 1999 through 2010 has been examined and in no single case has either of these “spare” bi-level values been reported.

In normal science operation, the BEP’s software housekeeping task calls its `Leds::show()` method (Figure 2) every 640 task interrupts (~64 seconds) to report 1STAT3ST, 1STAT2ST and 1STAT1ST, and to toggle the value of 1STAT0ST. `doLeds()` in turn calls `Leds::show()` to drive the hardware. It is a simple matter to insert code into this process to monitor threshold counts within the BEP and to force the four low-order bi-levels to the LED_BOOT_SPARE1 state.

The 16 combinations that can be assigned to the 4 bits under flight software control are shown in Table 2. Note that the BEP only uses the first 8 values during normal operations and 6 of the remainder when booting up, leaving two values completely unused. Chandra telemetry from 1999 through 2010 has been examined and in no single case has either of these “spare” bi-level values been reported.

Table 2. ACIS Flight Software Bi-Level Assignments

State Symbol	Bit Pattern	Instrument State
LED_WD_SCIENCE_A	x x x x 0 0 0 0	Most recent boot was from watchdog timer (patches not installed). Performing Science Run.
LED_WD_SCIENCE_B	x x x x 0 0 0 1	
LED_WD_IDLE_A	x x x x 0 0 1 0	Most recent boot was from watchdog timer (patches not installed). Not performing science run.
LED_WD_IDLE_B	x x x x 0 0 1 1	
LED_RUN_SCIENCE_A	x x x x 0 1 0 0	Most recent boot was commanded.
LED_RUN_SCIENCE_B	x x x x 0 1 0 1	Performing Science Run.
LED_RUN_IDLE_A	x x x x 0 1 1 0	Most recent boot was commanded.
LED_RUN_IDLE_B	x x x x 0 1 1 1	Not performing science run.
LED_RUN_STARTUP	x x x x 1 0 0 0	Task executive is starting up
LED_RUN_PATCH	x x x x 1 0 0 1	Resetting patch list
LED_BOOT_UPLINK_EXECUTE	x x x x 1 0 1 0	Calling loaded program
LED_BOOT_UPLINK_COPY	x x x x 1 0 1 1	Waiting for “Continue Upload” packets
LED_LBOOT_UPLINK_WAIT	x x x x 1 1 0 0	Waiting for “Start Upload” packet
LED_BOOT_SPARE1	x x x x 1 1 0 1	Unused: to be assigned to the ACIS radiation alert
LED_BOOT_SPARE2	x x x x 1 1 1 0	Unused
BOOT_RESET	x x x x 1 1 1 1	Software is starting up

Figure 2. The `Leds::show()` method

```
void Leds::show(unsigned value)
{
    bepReg.showLeds(value);
}
```

6. Patching the Flight Software

To determine where best to patch the flight software to accumulate the threshold counts, it is necessary to review the BEP architecture. Event records are read from the FEP-BEP ring buffers by the `processRecord()` methods of the `PmEvent`, `PmHist`, and `PmRaw` classes. Each calls `EventExposure::copyExpEnd()` to parse the `FEPexpEndRec` records that contain thresholds, the count of threshold crossings, and `expnum`, the exposure number, but this routine (see Figure 3) doesn't have access to the `ccdId` that labels the record and which will be needed to accumulate the crossings from that particular CCD.

Figure 3. The original `copyExpEnd()` method

```
void EventExposure::copyExpEnd(const FEPexpEndRec* dataptr)
{
    if ((expNum + 1) != dataptr->expnum) {
        swHousekeeper.report (SWSTAT_SCI_EXPEND_EXPNUM, dataptr->expnum);
    }
    expThresholdCnt = dataptr->thresholds;
    expParityErrs = dataptr->parityerrs;
}
```

Luckily, the MIPS CPU architecture makes it relatively easy to make inline patches that permit additional arguments to be passed to subroutines. In the current case, described in detail in Section 8 below, we shall patch the routines that call `copyExpEnd()` in order to pass an extra argument. When `processRecord()` is called with a `PmEvent` object, this argument will be the address of the object, but for other callers, *i.e.*, `PmRaw` or `PmHist` (*i.e.* raw frame or raw histogram mode), the argument will be null to show that these modes don't count threshold crossings. Since `PmEvent` is a sub-class of `ProcessMode`, the `ccdId` value can then be determined by a call to `getCcdId()`. A suitable replacement for `copyExpEnd()` is shown in Figure 4. It is called with an object of class `EventExposure`, and it calls `saveTXings()` with a static `TXings` object named `txings` (see Figure 5) in which the threshold crossing accumulators are to be stored.

Figure 4. Replacement for `copyExpEnd()`

```
void
Test_EventExposure::copyExpEnd(const FEPexpEndRec* dataptr, const ProcessMode* pm)
{
    if ((expNum + 1) != dataptr->expnum) {
        swHousekeeper.report (SWSTAT_SCI_EXPEND_EXPNUM, dataptr->expnum);
    }
    expThresholdCnt = dataptr->thresholds;
    expParityErrs = dataptr->parityerrs;

    // if called from PmEvent::processRecord(), save threshold counts
    if (pm != (ProcessMode*)0) {
        txings.saveTXings(pm->getMode(), pm->getCcdId(),
            dataptr->expnum, expFepTime, expThresholdCnt);
    }
}
```

Figure 5. The TXings class

```

#define private public
#include "filesscience/smtimedexposure.H"
#include "filesscience/smcontclocking.H"
#undef private
#include "filesscience/sciencemanager.H"
#include "filesswhouse/swhousekeeper.H"
#include "filesmemserver/memoryserver.H"

extern SmTimedExposure smTimedExposure;
extern SmContClocking smContClocking;
extern ScienceManager scienceManager;

class TXings {

public:
    void saveTXings(const ScienceMode* sm, unsigned ccdId,
                   unsigned expnum, unsigned fepTime, unsigned& thresh);
    Boolean triggerRadmon(void);

private:
    struct _TX { // parameter structure
        unsigned MINUTES; // averaging 64-sec intervals
        unsigned TRIGGER_COUNT; // threshold counter
        unsigned MAX_TX_PER_ROW; // max crossings per row
        unsigned CC_TICKS; // FEP ticks per frame in CC mode
        unsigned RATE_LIMIT[2]; // trigger thresholds/hundred-rows/sec
        unsigned TX_INCR[2]; // trigger threshold increments
    } TX;

    struct { // accumulator structure
        unsigned count; // count of calls to saveTXings
        Boolean triggered; // BoolTrue when alarm triggered
        unsigned minutes; // 64-second interval count
        unsigned ccd_rows; // number of CCD rows contributing
        unsigned ccd_tx_max; // max accepted crossings per row
        unsigned ccd_ticks; // frame readout time (10 usecs)
        unsigned increment; // additional crossings (test only)
        unsigned trigger_count[2]; // FI/BI intervals over threshold
        unsigned saved_rates[2]; // FI/BI rates
        unsigned threshold_accum[10]; // threshold accumulators
        unsigned exposure_accum[10]; // time tick accumulators
    } tx;
};

TXings txings; // a single static TXings object

// initialization for the TX structure
struct TXings::_TX TXinit = { 5, 5, 512, 291840, { 700, 40 }, { 8, 8 } };
struct TXings::_TX TXnext = { 5, 5, 512, 291840, { 700, 40 }, { 8, 8 } };

```

The `saveTXings()` method (see Figures 4 and 6) is called once for each event-mode exposure frame. The first time that it is called in a science run, it determines the number of read-out rows, the maximum anticipated number of non-pathological threshold crossings per frame, and the frame exposure time in units of the FEP pixel clock (*i.e.*, 10 μ s), and it increments the `tx.threshold_accum` and `tx.exposure_accum` accumulators. Integration times of less than 2000 seconds are guaranteed not to overflow either accumulator.

Since the number of rows per frame and the frame exposure time are constant in continuous clocking mode, they are initialized in the TX structure, but in timed-exposure mode, the frame time depends on the `dutyCycle`, `primaryExposure`, and `secondaryExposure` parameters. These are extracted from the external `pramTe` object, where they were copied from the science run parameter block when the run started.

Figure 6. The `saveTXings()` method

```

void TXings::saveTXings(const ScienceMode* sm, unsigned ccdId,
    unsigned expnum, unsigned fepTime, unsigned& thresh)
{
    // Check validity of arguments
    if (ccdId > 9 || tx.triggered == BoolTrue) {
        return;
    }

    // On new science run, reload TX parameters, clear accumulators
    if (tx.count++ == 0) {
        TX = TXnext;           // load parameters from TXnext
        TXnext = TXinit;      // load TXnext with defaults
        for (int ii = 1; ii < sizeof(tx)/sizeof(unsigned); ii++) {
            ((unsigned*)&tx)[ii] = 0;
        }
    }

    // Determine exposure time and row count
    if (TX.MINUTES == 0) {
        return;
    } else if (sm == (ScienceMode*)&smTimedExposure) {
        // Timed exposure mode
        if (pramTe.dutyCycle != 0 || tx.ccd_ticks == 0) {
            unsigned sw = (expnum % (pramTe.dutyCycle + 1)) != 0;
            tx.ccd_ticks = pramTe.exposureTime[sw] * 10000 + 4104;
            if (tx.ccd_rows == 0) {
                // First call in timed exposure mode
                tx.ccd_rows = pramTe.summedRows();
                tx.ccd_tx_max = (tx.ccd_rows * TX.MAX_TX_PER_ROW) >> pramTe.sumFlag;
            }
        }
    } else if (sm != (ScienceMode*)&smContClocking) {
        // unrecognized ScienceMode
        return;
    } else if (tx.ccd_ticks == 0) {
        // First call in continuous clocking mode
        tx.ccd_ticks = TX.CC_TICKS;
        tx.ccd_rows = pramCc.summedRows();
        tx.ccd_tx_max = (tx.ccd_rows * TX.MAX_TX_PER_ROW) >> pramCc.colSum;
    }

    // ignore zero or excessive crossings
    if (thresh > 0 && thresh <= tx.ccd_tx_max) {
        thresh += tx.increment;           // increment crossings only when testing
        tx.threshold_accum[ccdId] += thresh;
        tx.exposure_accum[ccdId] += tx.ccd_ticks;
    }
}

```

The radiation triggering algorithm is run in the `triggerRadmon()` routine (see Figure 7). It is called every 64 seconds whether or not a science run is in progress. If it isn't, `tx.count` is set to zero until a subsequent call to `saveTXings()` from `copyExpEnd()` reloads the TX parameter structure from `TXnext`.

Figure 7. The `triggerRadmon()` method

```

Boolean TXings::triggerRadmon(void)
{
    // if not running a science mode, clear the crossings count and trigger flag
    if (scienceManager.isIdle() == BoolTrue) {
        tx.count = 0;
        tx.triggered = BoolFalse;
    }

    // if already triggered, return immediately
    if (tx.triggered == BoolTrue) {
        return BoolTrue;
    }

    // Examine threshold crossings every TX.MINUTES*64 seconds
    if (tx.count == 0 || TX.MINUTES == 0 || tx.ccd_rows == 0 ||
        ++(tx.minutes) < TX.MINUTES) {
        return BoolFalse;
    }

    // Clear the counters (index tt = FI, BI)
    unsigned ccdcount[2] = { 0, 0 }; // number of CCDs of type ii
    unsigned ratecount[2] = { 0, 0 }; // number of CCDs of type ii reporting
    unsigned rateavg[2] = { 0, 0 }; // average count rate for type ii

    // Compute average threshold crossing rates for FI, BI separately
    for (unsigned cc = 0; cc < 10; cc++) {
        if (tx.exposure_accum[cc] > 0) {
            unsigned tt = ( cc == 5 || cc == 7 );
            unsigned exptime = ( tx.exposure_accum[cc] + 500 ) / 1000;
            unsigned rate = tx.threshold_accum[cc] / exptime;
            rate = ( 10000 * rate ) / tx.ccd_rows;
            ccdcount[tt]++;
            if (rate >= TX.RATE_LIMIT[tt]) {
                rateavg[tt] += rate;
                ratecount[tt]++;
            }
        }
    }

    // Test the average BI and FI chip rates separately
    for (unsigned tt = 0; tt < 2; tt++) {
        if (ratecount[tt] > 0 && ratecount[tt] == ccdcount[tt]) {
            unsigned rate = rateavg[tt] + ratecount[tt]/2;
            rate /= ratecount[tt];
            if (rate > tx.saved_rates[tt] + TX.TX_INCR[tt]) {
                tx.saved_rates[tt] = rate;
                if (++tx.trigger_count[tt] >= TX.TRIGGER_COUNT) {
                    tx.triggered = BoolTrue;
                }
                continue;
            }
        }
        tx.saved_rates[tt] = 0;
        tx.trigger_count[tt] = 0;
    }
}

```

continued overleaf...

Figure 7 (continued). The `triggerRadmon()` method

```

// If triggered, send a readBep packet and return
if (tx.triggered == BoolTrue) {
    unsigned *addr = (unsigned*)&TX;
    unsigned nword = (sizeof(TX)+ sizeof(tx))/sizeof(unsigned);
    CmdResult rc = memoryServer.readBep(1, addr, nword, TTAG_READ_BEP);
    if (rc != CMDRESULT_OK) {
        swHousekeeper.report(SWSTAT_CMDECHO_DROPPED, (unsigned)addr);
    }
    return BoolTrue;
}

// reset the accumulators and interval counter and return
for (unsigned id = 0; id < 10; id++) {
    tx.threshold_accum[id] = tx.exposure_accum[id] = 0;
}
tx.minutes = 0;
return BoolFalse;
}

```

In Section 5, we had talked about replacing `Leds::show()`. Since this routine is also called when booting up, we must be careful to not intercept calls until after the science manager has started (see Figure 8).

Figure 8. The `Test_Leds` class

```

class Test_Leds {
    void Test_Leds::show(unsigned value);
    {
        DebugProbe probe;

        // if the BEP is not booting up, check for a threshold trigger
        if (((value & 0x08) == 0 || (value & 0x0f) == LED_BOOT_SPARE1)
            && (txings.triggerRadmon() == BoolTrue)) {
            value = LED_BOOT_SPARE1;
        }
        bepReg.showLeds(value);
    }
};

```

7. Control Flow

After the `TXings` patch has been uploaded and the BEP warm-booted, the `tx.count` and `tx.triggered` fields will be initialized to zero by the patch loader. The first time an event-mode science run reads a `FEPexpEndRec` record from the FEP-BEP ring buffer, it will call `saveTXings()`, which will reinitialize the radiation filter parameters from the `TXnext` structure (see Figure 6). This makes it easy to change the filter parameters for subsequent science runs, as described in Section 9.

`Test_Leds::show()` calls `txings.triggerRadmon()` every 64 seconds to compute the average threshold crossing rates and look for triggers. When a trigger occurs, `triggerRadmon()` sets `tx.triggered` to `BoolTrue` and commands the memory manager thread to send a `bepReadReply` packet to telemetry, reporting the values of the `txings` parameters and variables. Then `Test_Leds::show()` sets the software bi-level channels to `LED_BOOT_SPARE1`, which persists for the remainder of the science run.

After the science run ends, the next call to `triggerRadmon()` from `Leds::show()` sets `tx.count` to zero and `tx.triggered` to `BoolFalse`, canceling the special bilevel value and preventing threshold crossing triggers until the next science task starts, calls `saveTXings()`, and reloads the `TX` structure.

8. Inline Patches

When they find a `FEPexpEndRec` record in the FEP-BEP ring buffer, the `processRecord()` methods of the three `ProcessMode` subclasses (`PmEvent`, `PmHist`, and `PmRaw`) call `copyExpEnd()` with assembler code fragments as shown in Figure 9. The `nop` instructions are added by the compiler because the `lw` (load from memory) instruction requires two machine cycles. A `nop` can be replaced by another machine instruction provided the latter doesn't address a register that is being loaded from—or stored into—external memory.

Figure 9. Assembler code segment when `PmEvent::processRecord()` calls `copyExpEnd()`

```
// getExposureInfo().copyExpEnd ((const FEPexpEndRec*) record.data);
...           // $2 = address of EventExposure object
lw   $3,84($2) // load address of EventExposure method table
nop           // wait for the value to appear in $3
lw   $3,32($3) // load address of copyExpEnd()
nop           // replace this instruction with "move $6,$17"
move  $4,$2    // arg1 = address of expInfo object
jal  $31,$3    // call copyExpEnd()
move  $5,$19   // arg2 = data address (executed before the jal jump)
```

If the second `nop` in Figure 9 is replaced by a `move $6,$17` instruction, the effect will be to pass the contents of register `$17` (which contains the address of the caller's `PmEvent` object) as a second argument to `copyExpEnd()`. By inspection, register `$6` is not used for any other purpose in the `processRecord()` callers, and by MIPS linkage convention, `$6` is never restored upon exiting a subroutine. For `PmHist` and `PmRaw`, we load `$6` with a zero value (`move $6,$0`) to signal that these modes do not measure threshold crossings. The inline patches (see Figure 10) are defined in a simple language that combines assembler instructions with names that define ranges of byte offsets from external address references.

Figure 10. Assembler patch for the three `copyExpEnd()` calls

```
.set noreorder
.set nomacro
.set noat
.text

#
# pass address of PmEvent object to Test_EventExposure::copyExpEnd()
#
.globl pmevent_lst_04d4_04d4
.ent pmevent_lst_04d4_04d4
pmevent_lst_04d4_04d4:
move $6,$17 # arg2 = address of caller's object
.end pmevent_lst_04d4_04d4

#
# pass null pointer to Test_EventExposure::copyExpEnd()
#
.globl pmhist_lst_01c0_01c0
.ent pmhist_lst_01c0_01c0
pmhist_lst_01c0_01c0:
move $6,$0 # arg2 = NUL
.end pmhist_lst_01c0_01c0

#
# pass null pointer to Test_EventExposure::copyExpEnd()
#
.globl pmraw_lst_0244_0244
.ent pmraw_lst_0244_0244
pmraw_lst_0244_0244:
move $6,$0 # arg2 = NUL
.end pmraw_lst_0244_0244
```

9. Operations

Once it is included in a patch load, and the BEP is warm-booted, the *txings* patch will be active during all subsequent science runs. When triggered by high and increasing threshold crossings, it sets the ACIS software bi-level values to `LED_BOOT_SPARE1` until the science run ends, or until the `tx.triggered` field is explicitly cleared by a *writeBep* command. This guarantees that it will appear in Chandra major frame readouts (once per 32.4 seconds).

The OBC should be patched to examine the ACIS bi-levels. It should safe the instruments if (a) `RADMON` is enabled, and (b) the bi-level channels (`1STAT3ST-1STAT0ST`) have the `LED_BOOT_SPARE1` values (1, 1, 0, 1).

The default filter parameters can be overridden by sending single *writeBep* command to ACIS to change the contents of the `TXinit` structure, whose address will depend on the ACIS flight software patch level (e.g., `0x8003dc30` in the current level E-F-G version). The command `"write 0 0x8003dc30 {\n0\n}"` will, for instance, suspend the threshold crossing filter, and `"write 0 0x8003dc30 {\n5\n}"` will turn it on again with an integration time of 5 minutes.

After a trigger, the bi-levels are not reset until `Leds::show()` is called when a science run is not in process. In the unlikely event that there is less than 64 seconds between the end of the triggering run and the start of the next, the bi-levels will continue to report `LED_BOOT_SPARE1`. This can be prevented by issuing a *writeBep* command to clear the counters: `"write 0 0x8003dc90 {\n0 0\n}"` prior to the second *startScience*.

In normal operation, most science runs can be conducted with *txings* enabled, but exceptionally bright targets observed by few CCDs may lead to false triggers. It might be best to disable *txings* for short runs where the risk of radiation damage is small, or turn on additional CCDs for longer runs to reduce the likelihood of a false trigger. To change the trigger parameters for the next science run only, a *writeBep* command should update the fields in `TXnext` rather than `TXinit`, and this must be done before the science run has started to report events. In the current level E-F-G version, `TXnext` is located at `0x8003dc50`.

When a threshold crossing trigger occurs, `triggerRadmon()` commands the BEP's memory manager to write a *bepReadReply* packet to telemetry, reporting the contents of the `TX` and `tx` structures. If this action is blocked for any reason, a `SWSTAT_CMDECHO_DROPPED` event will be reported in software housekeeping.

The current version of the patch reports *bepReadReply* packets with a *formatTag* of `TTAG_READ_BEP`. If this causes confusion, a new `TlmFormatTag` value could be defined, but the CXC Data System would need to be reconfigured to handle it. Similarly, if `SWSTAT_CMDECHO_DROPPED` is confusing, a new `SwStatistic` value could be defined.

10. Testing

Reliable automated tests of the *txings* patch have proved to be quite difficult to implement. In a first attempt, we built a stand-alone patch and ran it in the Engineering Unit in timed-exposure mode, commanding the Image Loader to write to the FEPs a series of data streams containing an increasing number of high-valued pixels. This was sufficient to test the patch itself, but frequently caused a "T-plane Latch-Up" in one or more of the FEPs. This indicates that the pixel processor in an Actel FPGA had halted, and was likely caused by a known design error in the Actel's microcode. This problem first showed up prior to launch when a FEP's firmware had been stressed by simultaneously (a) processing many threshold crossings per frame, (b) making many memory accesses from the FEP CPU, and (c) handling frequent remote memory accesses through the FEP-BEP interface. We updated BEP flight software to prevent this from occurring in flight, and it has only occurred on a few occasions when, for unknown reasons, the BEP began copying FEP bias maps during event processing. Its recurrence when handling modest numbers of event crossings and event candidates was unexpected. Assuming that it was our reloading the images during event processing that was causing the latch-ups, we changed the testing strategy, commanding the Image Loader to write a single image frame containing a variety of pixel values, but varying the FEP thresholds by sending a series of *writeFep* commands to the BEP to update the thresholds in the `FEPparamBlock` structures in the FEPs' D-caches. This had no noticeable effect on the probability of latch-ups.

Figure 11. Extract from *timed-exposure event-processing tests in runtst.tcl*

```

# ---- Load TX addresses in BEP ----
source "txings.def"

# ---- Make the Bias Run ----
system make biaste ROWS=$rows
send -i $cmd_id "start 0 te bias 4\n"
command_echo 1 15 "start bias run"
wait_stop_science

# ---- Load the TX parameter block ----
send -i $cmd_id "write 0 $txnext {\n 3 3 512 145920 700 40 8 8 \n}\n"
command_echo 1 192 "write TX block"

# ---- Start the Science Run ----
system make imagete ROWS=$rows
send -i $cmd_id "start 0 te 4\n"
command_echo 1 14 "start science run"

# ---- Conduct the Run ----
set state 0
set inc 0
set timeout 360
expect {
    -re "swHousekeeping\[\r\]*\[\r\n\]+" {
        if {$state == 1} {
            send -i $cmd_id "write 0 $txincraddr {\n $inc \n}\n"
            incr inc 500
            send -i $cmd_id "wait 1\nread 0 $txblock $txlen\n"
        }
        exp_continue
    }
    -re ".*SWSTAT_FEP_STARTDATA:\[\r\]*\[\r\n\]+" {
        set state 1
        exp_continue
    }
    -re "bepReadReply\[\r\]*commandId=1 \[\r\]*\[\r\n\]+" {
        set state 2
    }
    -re "scienceReport\[\r\]*terminationCode=(\[0-9]+\)\[\r\n\]+" {
        fail "BEP termination code $expect_out(1,string)"
    }
    timeout { fail "Timeout: no RADMON bepReadReply state $state" }
}

# ---- Wait for bilevels to be reported ----
if {$state == 2} {
    set psci_id $spawn_id
    spawn /bin/sh -c "filterClient -h $env(ACISSERVER) | ltlm -p61 -v"
    expect {
        -re "value *= \[0-9]+\ \#\ \{(1011.*\[\r\n\]*" { set state 2 }
        timeout { }
    }
    set spawn_id $psci_id
}
}

```


We therefore tried a third strategy: leaving the input stream and the FEPs alone and changing the *txings* patch so that it increases the apparent threshold count itself. As shown in Figure 6, `saveTXings()` increments its caller's `thresh` variable by the value of `tx.increment`. The latter field will be zero in flight, but for testing purposes it is set to successively higher values by a series of *writeBep* commands, as shown in Figure 11, where it is addressed as *\$txincraddr* (0x8003dca8 in the E-F-G patch load).

Patch-dependent addresses are defined in file *"txings.def"*. For testing purposes, the default `TX` parameters were overridden so as to shorten the run time and to decrease the threshold levels, reducing the possibility of a FEP latch-up. A bias image was written to the Image Loader and a bias-only run made. Then a test image was sent to the Image Loader and a science run started. After every software housekeeping packet was telemetered—at intervals of ~64 secs—a *writeBep* was issued to add 500 to `tx.increment` and the contents of the `txings` object was then read to telemetry with a *readBep* command. This continued until the radiation alert was triggered (or until the run terminated abnormally or timed out). The `expect` loop exited and a second one was started to look for the "1101" bilevel values indicating that the patch had reported correctly.

The *"runtest.tcl"* script (see Figure 11) conducts 6 separate tests: timed-exposure mode (full-frame, and event histogram), continuous clocking mode (3x3 and 1x3), and continuous and timed raw mode (see Figure 12). The relevant patches are loaded at the start of the script, after which the tests are run without rebooting the BEP or power-cycling the FEPs.

Figure 12. Extract from a raw-mode test in *runtest.tcl*

```

set timeout 360
set state 0
expect {
    -re "data..Raw\[^\r]*\[\r\n]+" {
        if {$state == 0} {
            send -i $cmd_id "stop 0 science\n"
            set state 1
        }
        exp_continue
    }
    -re "swHousekeeping\[^\r]*\[\r\n]+" {
        if {$state == 1} {
            send -i $cmd_id "read 0 $txblock $txlen\n"
            set state 2
        }
        exp_continue
    }
    -re "bepReadReply\[^\r]*commandId=(\[0-9])\[^\r]*\[\r\n]+" {
        if {$expect_out(1,string) != 2} {
            fail "Bad bepReadReply id=$expect_out(1,string)"
        }
        exp_continue
    }
    -re "scienceReport\[^\r]*terminationCode=(\[0-9]+)\[\r\n]+" {
        if {$expect_out(1,string) != 1} {
            fail "BEP termination code $expect_out(1,string)"
        }
    }
    timeout { fail "Timeout: no RADMON bepReadReply state $state" }
}

# ---- Check the TX Block against txings.txt ----
set cmd "perl ltlm -X -p1 -v pkts.raw | tail -24 | diff - txings.txt"
if {[catch {system $cmd} err]} {
    fail $err
}

```

Table 3. Event threshold values used in *runtest.tcl*

Test Mode	Front-Illuminated CCDs			Back-Illuminated CCDs		
	Threshold (ADU)	Threshold Crossings	Event Candidates	Threshold (ADU)	Threshold Crossings	Event Candidates
Timed Exposure (3x3)	580	12584	108	665	2409	21
Continuous Clocking (3x3)	285	7260	63	330	1936	17
Continuous Clocking (1x3)	320	3146	297	335	1210	121

The FEP thresholds and resulting numbers of threshold crossings and event candidates used in *runtest.tcl* are shown in Table 3. The threshold crossing count used by *txings* is augmented in steps of 500 ADU at 64 second intervals.

A lingering doubt remained. When the EU FEPs reported a sustained rate of more than $\sim 10,000$ threshold crossings, passing several hundred event candidates per frame to the BEP, there was a high likelihood that the FEP’s pixel processing firmware would fail, either by latching the T-plane or by advancing the exposure counter by large increments for each frame VSYNC received from the Image Loader. This behavior has never been seen on the flight unit, and appears unrelated to the *txings* patch—it occurs as frequently when the patch is removed—and may point to a hitherto unrecognized problem with the Image Loader-to-DPA interface.

To better validate the patch, we decided to add an additional set of tests, *runtest2.tcl*, that by-pass the Image Loader and use the EU DEA as a source of pixels. With resistive loads across their analog inputs, the DEAs output a pixel stream from each CCD quadrant, whose 12-bit values are well approximated by Gaussian functions with FWHM of 5–10 ADU. By choosing suitable values of *eventThreshold* and *videoOffset* in the timed-exposure and continuous-clocking parameter blocks, the average threshold crossing rates per frame can be “tuned” to lie within the 10,000–20,000 required to test the *txings* patch. However, this gives rise to an unwanted side effect: since each above-threshold pixel is most likely to be a local maximum, the number of event candidates reported by each FEP will be almost as large as the number of threshold crossings, and will exceed the number than can be processed by the BEP during a single exposure time. This causes the FEPs to drop exposures—typically 4 out of every 5—and we are no longer testing a flight-like scenario. We have therefore created a new *fepthrrottle* patch (see Figure 13) whose sole function is to reduce the number of event candidates reported by each FEP by a factor of ~ 100 . This patch is only to be used in conjunction with the *txings* patch, and only on the Engineering Unit, and is not therefore added to the *release-E-opt-F* package.

Figure 13. The *throttleFepAppendRingBuf* routine

```
#include "fepCt1.h"

void throttleFepAppendRingBuf(unsigned *ptr, unsigned wordcnt, FEpparm *fp)
{
    unsigned fepthrrottle = 100;
    if ((ptr[1] % fepthrrottle) == 0) { /* ptr[1] is "short row,col" */
        fepAppendRingBuf(ptr, wordcnt, fp);
    }
}
```

The *fepthrrottle* patch also includes inline patches to call `throttleFepAppendRingBuf()` in place of `fepAppendRingBuf()` from within the event-processing routines in *sepSciTimed.c* and *sepSciCClk.c*, and from within the *cc3x3* patch. The test script, *runtest2.tcl*, is essentially the same as *runtest.tcl* (see Figures 12 and 13), with the omission of the bias -only runs—the bias maps can be created directly from the DEA noise input. The event thresholds and video offsets (4 nodes per CCD) as listed in Table 4, alongside the resulting average counts of threshold crossings and event candidates. The latter are suppressed relative to their “true” values by means of the *fepthrrottle* patch.

Table 4. Event threshold and video offset values used in *runtest2.tcl*

FEP	CCD	Event Thresholds	Video Offsets	Threshold Crossings			Event Candidates		
				TE	CC3x3	CC1x3	TE	CC3x3	CC1x3
0	S3	7, 7, 7, 7	33, 38, 38, 38	12750	2722	4663	107	12	31
1	I0	7, 7, 7, 7	33, 33, 33, 33	13296	3152	3835	115	16	22
2	I1	4, 4, 4, 4	43, 44, 43, 23	10709	2908	3131	88	13	15
3	I2	4, 4, 4, 4	33, 33, 33, 33	8768	4752	2568	68	30	10
4	I3	6, 6, 6, 6	33, 33, 33, 33	19080	4371	3960	169	27	23
5	S1	7, 7, 7, 7	33, 33, 33, 33	12555	3194	3208	108	16	16

11. References

- “Using ACIS on the Chandra X-ray Observatory as a particle radiation monitor,” C. E. Grant, B. LaMarr, M. W. Bautz and S. L. O’Dell, SPIE, June 2010.
- “DPA Hardware Specification and System Description,” MIT 36-02104, Rev. C, April 15, 1997.
- “ACIS Software User’s Guide,” MIT 36-54003, Rev. A, (NAS8-37716/DR/SDM05) July 21, 1999.
- “ACIS Software Detailed Design Specification (As-Built),” MIT 36-53200, Rev. A, (NAS8-37716/DR/SDM03) February 3, 2000.

12. Glossary

- 1STAT n S The software names for the 8 one-bit ACIS bilevel fields ($n = 0..7$).
- ACISSERVER UNIX Environment variable containing the host name of the EU interface.
- ACTEL A manufacturer’s brand name of FPGA used in BEPs and FEPs.
- ADU Analog Data Unit — the least significant bit of a 12-bit pixel value.
- Back-Illuminated A CCD that detects x-rays incident on the face opposite to that of its junctions.
- BEP ACIS Back End Processor — the unit that interfaces between RCTU and FEPs.
- BI An abbreviation for Back-Illuminated. (*q.v.*)
- Bi-Level A data channel from DPA to RCTU reporting only OFF or ON.
- Bias The average value for a pixel that contains no event or background charge.
- BoolFalse The “false” value for a software boolean field.
- BoolTrue The “true” value for a software boolean field.
- CC An abbreviation for Continuous Clocking (*q.v.*)
- CC1x3 Continuous Clocking mode that reports 1 (row) by 3 (columns) events.
- CC3x3 Continuous Clocking mode that reports 3 (rows) by 3 (columns) events.
- CCD Charge-Coupled Device — the x-ray detectors used by ACIS.
- ccdId The index of a particular ACIS CCD (0..3 = I0..I3, 4..9 = S0..S5).
- CmdResult The result returned by a BEP command (1 = success).
- CPU Central Processing Unit — ACIS FEPs and BEPs use the R3000 *Mongoose*.
- CXC Chandra X-Ray Observatory Science Center.
- D-cache The radiation-hard data cache memory used in BEPs and FEPs.
- DC Direct Current — the average zero-event input to the DEAs.
- DEA ACIS Detector Electronics Assembly — CCD sequencers and digital converters.
- DPA ACIS Digital Processor Assembly — containing 6 FEPs and 2 BEPs.
- E1300 EPHIN channel most sensitive to low-energy protons that can damage ACIS.
- EPHIN Electron, Proton and Helium Instrument — flown on Chandra and SOHO.
- EU ACIS Engineering Unit — duplicate DEA+DPA+PSMC in an MIT laboratory.
- EventExposure BEP software object containing details of FEP event candidates.
- FEP ACIS Front End Processor — extracts event candidates from a pixel stream.

FEP-BEP	The memory-mapped interface between the BEPs and the 6 FEPs.
FEPexpEndRec	FEP-BEP record indicating the end of an exposure frame.
FEPparm	FEP software structure containing all current variables.
FEPparmBlock	Software structure passed from BEP to FEP to control a science run.
FI	An abbreviation for Front-Illuminated. (<i>q.v.</i>)
FIFO	First-In-First-Out — the order in which BEP commands are processed.
FPGA	Field-Programmable Gate Array — in ACIS BEPs and FEPs (see ACTEL).
Front-Illuminated	A CCD that detects x-rays incident on the same face as its junctions.
FWHM	Full Width at Half Maximum — the width of a Gaussian distribution.
I-cache	The radiation-hard instruction cache memory used in BEPs and FEPs.
Inline	A patch that replaces existing code without requiring additional storage.
Latch-Up	What happens when all or part of a FEP ACTEL stops working.
LED_BOOT_SPARE1	The 4-bit value of 1STAT3ST–1STAT0ST indicating a radiation threshold trip.
Leds	Light-Emitting Diodes — original ACIS software name for bi-levels.
MIPS	Microprocessor without Interlocked Pipeline Stages — a.k.a. <i>Mongoose</i> CPU.
NUL	Zero value.
OBC	On-Board Computer — the Chandra spacecraft’s central controller.
OBSID	Observation Identifier — a unique number assigned by the CXC to a science run.
PmEvent	BEP software class describing FEP event candidates.
PmHist	BEP software class describing FEP raw histograms.
PmRaw	BEP software class describing FEP raw frames.
ProcessMode	BEP software class describing the current TE or CC science mode.
PSMC	ACIS Power Supply and Mechanism Controller.
RADMON	Excessive radiation alert signal from EPHIN or ground to OBS and/or ACIS.
Raw Mode	ACIS processing mode that returns all pixel values.
RCTU	Remote Command and Telemetry Unit — interface between ACIS and the S/C.
readBep	External command to ACIS to dump specified contents of BEP memory.
readFep	External command to ACIS to dump specified contents of FEP memory.
ScienceManager	BEP software task and class to control science runs.
ScienceMode	BEP software class to control mode-dependent processing.
SCS107	Command to OBC from EPHIN or ground to safe the science instruments.
SmContClocking	Sub-class of ScienceMode to control Continuous Clocking runs.
SmTimedExposure	Sub-class of ScienceMode to control Timed Exposure runs.
SwHousekeeper	BEP software task and class to report software events at 64 second intervals.
SwStatistic	A field in SwHousekeeper telemetry packets to report event characteristics.
T-plane	Threshold Crossing Plane — 1-bit FEP memory recording threshold crossings.
TE	An abbreviation for Timed Exposure (<i>q.v.</i>)
Te3x3	Timed-Exposure mode that reports 3 (rows) by 3 (columns) events.
Te5x5	Timed-Exposure mode that reports 5 (rows) by 5 (columns) events.
Threshold	Value by which a pixel exceeds its corresponding bias value to become interesting.
TlmFormatTag	The field whose value distinguishes the possible types of ACIS telemetry packets.
TX	The sub-array in TXblock that is re-initialized at the start of a science run.
TXblock	The instance variables used by the <i>txings</i> patch during the current science run.
TXinit	The default set of TX parameters copied to TXnext at the start of a science run.
TXnext	The set of TX parameters copied to TX at the start of a science run.
tx	The sub-array in TXblock containing accumulators for the current science run.
VSYNC	Flag in the FEP pixel input stream marking the start of a new exposure frame.
writeBep	External command to ACIS to update specified contents of BEP memory.
writeFep	External command to ACIS to update specified contents of FEP memory.

```
#define private public
#include "filesscience/smtimedexposure.H"
#include "filesscience/smcontclocking.H"
#undef private
#include "filesscience/sciencemanager.H"
#include "filesswhouse/swhousekeeper.H"
#include "filesmemserver/memoryserver.H"

extern SmTimedExposure smTimedExposure;
extern SmContClocking smContClocking;
extern ScienceManager scienceManager;

// -----
// Class TXings -- where all the work is done
// -----

class TXings {
public:
    void saveTXings(const ScienceMode* sm,
        unsigned ccdId, unsigned expnum, unsigned& thresh);
    Boolean triggerRadmon(void);
private:
    struct _TX {
        unsigned MINUTES;           // averaging interval x 64 seconds
        unsigned TRIGGER_COUNT;     // threshold counter
        unsigned MAX_TX_PER_ROW;    // max crossings per row
        unsigned CC_TICKS;         // ticks per frame in CC mode
        unsigned RATE_LIMIT[2];    // trigger thresholds/hundred-rows/sec
        unsigned TX_INCR[2];       // trigger threshold increments
    } TX;
    struct {
        unsigned count;            // number of calls to saveTXings
        Boolean triggered;         // true when alarm triggered
        unsigned minutes;         // 64-second interval count
        unsigned ccd_rows;        // number of CCD rows contributing
        unsigned ccd_tx_max;      // max accepted crossings per row
        unsigned ccd_ticks;       // CCD readout time (10 usecs)
        unsigned increment;       // additional crossings (test only)
        unsigned trigger_count[2]; // intervals over threshold
        unsigned saved_rates[2];   // BI and FI rates
        unsigned threshold_accum[10]; // threshold accumulators
        unsigned exposure_accum[10]; // time tick accumulators
    } tx;
};

TXings txings;           // single static TXings object

struct TXings::_TX TXinit = { 5, 5, 512, 291840, { 700, 40 }, { 8, 8 } };
struct TXings::_TX TXnext = { 5, 5, 512, 291840, { 700, 40 }, { 8, 8 } };

// -----
// TXings::saveTXings -- save event-mode threshold crossing counts
// -----

void TXings::saveTXings(const ScienceMode* sm,
    unsigned ccdId, unsigned expnum, unsigned& thresh)
{
    // Check validity of arguments
    if (ccdId > 9 || tx.triggered == BoolTrue) {
        return;
    }

    // on new science run, reload TX parameters, clear tx accumulators
    if (tx.count++ == 0) {
```

```
TX = TXnext;
TXnext = TXinit;
for (int ii = 1; ii < sizeof(tx)/sizeof(unsigned); ii++) {
    ((unsigned *)&tx)[ii] = 0;
}

// get rows and exposure time
if (TX.MINUTES == 0) {
    return;
} else if (sm == (ScienceMode*)&smTimedExposure) {
    if (pramTe.dutyCycle != 0 || tx.ccd_ticks == 0) {
        unsigned sw = (expnum % (pramTe.dutyCycle + 1)) != 0;
        tx.ccd_ticks = pramTe.exposureTime[sw] * 10000 + 4104;
        if (tx.ccd_rows == 0) {
            tx.ccd_rows = pramTe.summedRows;
            tx.ccd_tx_max =
                (tx.ccd_rows * TX.MAX_TX_PER_ROW) >> pramTe.sumFlag;
        }
    }
} else if (sm != (ScienceMode*)&smContClocking) {
    return;
} else if (tx.ccd_ticks == 0) {
    tx.ccd_ticks = TX.CC_TICKS;
    tx.ccd_rows = pramCc.summedRows;
    tx.ccd_tx_max = (tx.ccd_rows*TX.MAX_TX_PER_ROW) >> pramCc.colSum;
}

// ignore zero excessive crossings
if (thresh > 0 && thresh <= tx.ccd_tx_max) {
    thresh += tx.increment; // increment crossings only when testing
    tx.threshold_accum[ccdId] += thresh;
    tx.exposure_accum[ccdId] += tx.ccd_ticks;
}
}

// -----
// TXings::triggerRadmon -- determine whether to trigger RADMON
// -----

Boolean TXings::triggerRadmon(void)
{
    // if not running a science mode, clear the crossings count
    if (scienceManager.isIdle() == BoolTrue) {
        tx.count = 0;
        tx.triggered = BoolFalse;
    }

    // If already triggered, return immediately
    if (tx.triggered == BoolTrue) {
        return BoolTrue;
    }

    // Examine threshold crossings every TX.MINUTES*64 seconds
    if (tx.count == 0 || TX.MINUTES == 0 || tx.ccd_rows == 0 ||
        ++(tx.minutes) < TX.MINUTES) {
        return BoolFalse;
    }

    // clear the counters (index ii = FI, BI)
    unsigned ccdcount[2] = { 0, 0 }; // number of CCDs of type ii
    unsigned ratecount[2] = { 0, 0 }; // number of CCDs above threshold
    unsigned rateavg[2] = { 0, 0 }; // average count rate for type ii
}
```

```
// compute average threshold crossing rates
for (unsigned cc = 0; cc < 10; cc++) {
    if (tx.exposure_accum[cc] > 0) {
        unsigned tt = ( cc == 5 || cc == 7 );
        unsigned exptime = ( tx.exposure_accum[cc] + 500 ) / 1000;
        unsigned rate = tx.threshold_accum[cc] / exptime;
        rate = ( 10000 * rate ) / tx.ccd_rows;
        ccdcount[tt]++;
        if (rate >= TX.RATE_LIMIT[tt]) {
            rateavg[tt] += rate;
            ratecount[tt]++;
        }
    }
}

// test BI and FI chip rates separately
for (unsigned tt = 0; tt < 2; tt++) {
    if (ratecount[tt] > 0 && ratecount[tt] == ccdcount[tt]) {
        unsigned rate = rateavg[tt] + ratecount[tt]/2;
        rate /= ratecount[tt];
        if (rate > tx.saved_rates[tt] + TX.TX_INCR[tt]) {
            tx.saved_rates[tt] = rate;
            if (++tx.trigger_count[tt] >= TX.TRIGGER_COUNT) {
                tx.triggered = BoolTrue;
            }
            continue;
        }
    }
    tx.saved_rates[tt] = 0;
    tx.trigger_count[tt] = 0;
}

// if triggered, send a readBep packet and return
if (tx.triggered == BoolTrue) {
    unsigned *addr = (unsigned*)&TX;
    unsigned nword = (sizeof(TX)+sizeof(tx))/sizeof(unsigned);
    CmdResult rc = memoryServer.readBep(1, addr, nword, TTAG_READ_BEP);
    if (rc != CMDRESULT_OK) {
        swHousekeeper.report (SWSTAT_CMDECHO_DROPPED, (unsigned)addr);
    }
    return BoolTrue;
}

// reset the accumulators and return
for (unsigned id = 0; id < 10; id++) {
    tx.threshold_accum[id] = tx.exposure_accum[id] = 0;
}
tx.minutes = 0;
return BoolFalse;
}

// -----
// Class Test_EventExposure -- save event-mode threshold crossing counts
// -----

class Test_EventExposure : public EventExposure
{
public:
    void Test_EventExposure::copyExpEnd(const FEPexpEndRec* dataptr,
        ProcessMode* pm)
    {
        if ((expNum + 1) != dataptr->expnum) {
            swHousekeeper.report (SWSTAT_SCI_EXPEND_EXPNUM, dataptr->expnum);
        }
    }
};
```

```
expThresholdCnt = dataptr->thresholds;
expParityErrs = dataptr->parityerrs;

// if called from PmEvent::processRecord(), save threshold counts
if (pm != (ProcessMode*)0) {
    txings.saveTXings(pm->getMode(), pm->getCcdId(),
        dataptr->expnum, expThresholdCnt);
}
};

// -----
// Class Test_Leds -- call triggerRadmon() and maybe set bi-levels
// -----

class Test_Leds
{
public:
    void Test_Leds::show(unsigned value)
    {
        DebugProbe probe;

        // if the BEP is not booting up, test for RADMON
        if (((value & 0x08) == 0 || (value & 0x0f) == LED_BOOT_SPARE1)
            && (txings.triggerRadmon() == BoolTrue)) {
            value = LED_BOOT_SPARE1;
        }
        bepReg.showLeds(value);
    }
};

// -----
// End of txings patch
// -----
```



```
/*=====
//
// $Source: /acis/h3/acisfs/configctl/patches/txings/txingsinline.S,v $
//
// MODULE NAME:
//
// PURPOSE:
//
// REFERENCES:
//
// $Log: txingsinline.S,v $
// Revision 1.1 2010/11/17 20:45:49  pgf
// Initial version.
//
// COPYRIGHT: Massachusetts Institute of Technology 2008
//
=====*/

.set noreorder
.set nomacro
.set noat
.text

#####
#
# pass address of PmEvent object to Test_EventExposure::copyExpEnd()
#
#####

.globl pmevent_lst_04d4_04d4
.ent pmevent_lst_04d4_04d4
pmevent_lst_04d4_04d4:
move $6,$17
.end pmevent_lst_04d4_04d4

#####
#
# pass null pointer to Test_EventExposure::copyExpEnd()
#
#####

.globl pmhist_lst_01c0_01c0
.ent pmhist_lst_01c0_01c0
pmhist_lst_01c0_01c0:
move $6,$0
.end pmhist_lst_01c0_01c0

#####
#
# pass null pointer to Test_EventExposure::copyExpEnd()
#
#####

.globl pmraw_lst_0244_0244
.ent pmraw_lst_0244_0244
pmraw_lst_0244_0244:
move $6,$0
.end pmraw_lst_0244_0244
```

```
#! /usr/bin/env expect
#
# $Source: /nfs/acis/h3/acisfs/confignt1/patches/txings/testsuite/smoke/runtest.tcl,v $
#
# Tests of txings patch
#

send_user "Welcome to txings/testsuite/smoke/runtest.tcl\n"

# ---- Launch the command and telemetry server processes ----
set basedir [lindex $argv 0] ; # Patch base directory
set tools [lindex $argv 1] ; # Tool directory
set patchdir [lindex $argv 2] ; # Patch directory

# ---- Run Parameters ----
set ccd_list "7 0 1 2 3 5" ; # Desired fepCcdSelect
set rows 1024 ; # Number of rows in TE mode
set primary 32 ; # TE mode exposure time
set txminutes 3 ; # TX integration Time
set timeout 30 ; # Default timeout

# ---- Embed the Procedure Library ----
source $basedir/$tools/lib/lib-exp/runtest_support.tcl

# ---- Start the Command Pipe ----
spawn $basedir/$tools/bin/cmdclient $env(ACISSERVER)
set cmd_id $spawn_id

# ---- Start the Telemetry Pipe ----
spawn $basedir/$tools/bin/tlmclient $env(ACISSERVER)
sleep 1

# ---- Load TX Parameters ----
source "txings.def"

# ---- Halt the Processor ----
cold_boot

# ---- Load Patches ----
load_patch_list "$basedir/$tools/share/standard.bcmod\
    $basedir/$tools/share/opt_smtimedlookup.bcmod\
    $basedir/$tools/share/opt_eventhist.bcmod\
    $basedir/$tools/share/opt_cc3x3.bcmod\
    $basedir/$tools/share/opt_tlmio.bcmod\
    $basedir/$tools/share/opt_printshouse.bcmod\
    $basedir/$tools/share/opt_dearepl.bcmod\
    $basedir/$patchdir/opt_txings.bcmod"

# ---- Apply Patches ----
warm_boot

# ---- Power Down all Boards ----
power_off_boards
set timeout 10
expect { timeout {} }

# ---- Wait for FEPs to finish powering ----
for {set fep 5} {$fep >= 0} {incr fep -1} {
    if {[lindex [split $ccd_list " "] $fep] < 10} { break }
}
set timeout 120
power_on_boards "$ccd_list"
expect {
    -re ".*SWSTAT_FEP_EXECMEM: $fep\[\r\n]" { }
}
```

```
    timeout { fail "boards not powered on" }
}

# ---- Initialize the TX Blocks ----
proc load_tx_block {} {
    global cmd_id txnext txblock txlen txminutes

    # ---- Load the TXnext parameter block ----
    send -i $cmd_id "write 0 $txnext {\n $txminutes 3 512 145920 700 40 \n}\n"
    command_echo 1 192 "write TXnext block"

    # ---- Clear the TX block ----
    send -i $cmd_id "write 0 $txblock {\n"
    for {set i 0} {$i < $txlen} {incr i} {send -i $cmd_id "0\n" }
    send -i $cmd_id "}\n"
    command_echo 1 192 "clear TX counters"
}

# ---- Load TE Parameter Block ----
proc load_te_test { fepmode bepmode rows primary thFI thBI } {
    global cmd_id ccd_list

    # --- Send Parameter Block ---
    send -i $cmd_id "load 0 te 4 {
parameterBlockId          = 0xffffffff
fepCcdSelect              = $ccd_list
fepMode                   = $fepmode
bepPackingMode            = $bepmode
onChip2x2Summing          = 0
ignoreBadPixelMap         = 1
ignoreBadColumnMap        = 1
recomputeBias             = 0
trickleBias               = 0
subarrayStartRow          = 0
subarrayRowCount          = [expr $rows - 1]
overclockPairsPerNode     = 8
outputRegisterMode        = 0
ccdVideoResponse          =      0      0      0      0      0      0
primaryExposure           = $primary
secondaryExposure         = 0
dutyCycle                 = 0
fep0EventThreshold        = $thBI $thBI $thBI $thBI
fep1EventThreshold        = $thFI $thFI $thFI $thFI
fep2EventThreshold        = $thFI $thFI $thFI $thFI
fep3EventThreshold        = $thFI $thFI $thFI $thFI
fep4EventThreshold        = $thFI $thFI $thFI $thFI
fep5EventThreshold        = $thBI $thBI $thBI $thBI
fep0SplitThreshold        =      13      13      13      13
fep1SplitThreshold        =      13      13      13      13
fep2SplitThreshold        =      13      13      13      13
fep3SplitThreshold        =      13      13      13      13
fep5SplitThreshold        =      13      13      13      13
fep4SplitThreshold        =      13      13      13      13
fep5SplitThreshold        =      13      13      13      13
lowerEventAmplitude       = 2000
eventAmplitudeRange       = 10
gradeSelections           = 0xfeffffff 0xffffffff 0xffffffffb 0xfffff7ff\
                          0xffffffff 0xffffffff 0xffbffffff 0x7fffffff

windowSlotIndex           = 65535
histogramCount            = 10000
biasCompressionSlotIndex  =      1      1      1      1      1
rawCompressionSlotIndex  = 0
ignoreInitialFrames       = 2
biasAlgorithmId           =      1      1      1      1      1
    }
```

```
biasArg0          = 1 1 1 1 1 1
biasArg1          = 0 0 0 0 0 0
biasArg2          = 0 0 0 0 0 0
biasArg3          = 50 50 50 50 50 50
biasArg4          = 20 20 20 20 20 20
fep0VideoOffset  = 65 65 65 65
fep1VideoOffset  = 65 65 65 65
fep2VideoOffset  = 65 65 65 65
fep3VideoOffset  = 65 65 65 65
fep4VideoOffset  = 65 65 65 65
fep5VideoOffset  = 65 65 65 65
deaLoadOverride  = 0
fepLoadOverride  = 0\n}\n"

# ---- Wait for Echo ----
command_echo 1 9 "load te"
}

# ---- Load CC Parameter Block ----
proc load_cc_test { fepmode bepmode thFI thBI } {
    global cmd_id ccd_list

    # --- Send Parameter Block ---
    send -i $cmd_id "load 0 cc 4 {
parameterBlockId = 0xffffffff
fepCcdSelect     = $ccd_list
fepMode         = $fepmode
bepPackingMode  = $bepmode
ignoreBadColumnMap = 1
recomputeBias   = 0
trickleBias     = 0
rowSum          = 0
columnSum       = 0
overclockPairsPerNode = 8
outputRegisterMode = 0
ccdVideoResponse = 0 0 0 0 0 0
fep0EventThreshold = $thBI $thBI $thBI $thBI
fep1EventThreshold = $thFI $thFI $thFI $thFI
fep2EventThreshold = $thFI $thFI $thFI $thFI
fep3EventThreshold = $thFI $thFI $thFI $thFI
fep4EventThreshold = $thFI $thFI $thFI $thFI
fep5EventThreshold = $thBI $thBI $thBI $thBI
fep0SplitThreshold = 13 13 13 13
fep1SplitThreshold = 13 13 13 13
fep2SplitThreshold = 13 13 13 13
fep3SplitThreshold = 13 13 13 13
fep5SplitThreshold = 13 13 13 13
fep4SplitThreshold = 13 13 13 13
fep5SplitThreshold = 13 13 13 13
lowerEventAmplitude = 2000
eventAmplitudeRange = 10
gradeSelections     = 0x0e
windowSlotIndex    = 65535
rawCompressionSlotIndex = 0
ignoreInitialFrames = 2
biasAlgorithmId    = 1 1 1 1 1 1
biasRejection      = 384 384 384 384 384 384
fep0VideoOffset    = 65 65 65 65
fep1VideoOffset    = 65 65 65 65
fep2VideoOffset    = 65 65 65 65
fep3VideoOffset    = 65 65 65 65
fep4VideoOffset    = 65 65 65 65
fep5VideoOffset    = 65 65 65 65
deaLoadOverride    = 0
```

```
fepLoadOverride          = 0\n}\n"

# ---- Wait for Echo ----
command_echo 1 10 "load cc"
}

# ---- Run Event Mode Test ----
proc run_event_test { mode rows bias start } {
    global cmd_id spawn_id env txblock txlen txincraddr
    set timeout 360
    set state 0
    set inc 0

    # ---- Load the Bias Image ----
    system make bias$mode ROWS=$rows

    # ---- Load the TX parameter block ----
    load_tx_block

    # ---- Compute Bias Maps ----
    send -i $cmd_id "start 0 $mode bias 4\n"
    command_echo 1 $bias "start bias run"
    wait_stop_science

    # ---- Load Image ----
    system make image$mode ROWS=$rows

    # ---- Load the TX parameter block ----
    load_tx_block

    # ---- Start the Science Run ----
    send -i $cmd_id "start 0 $mode 4\n"
    command_echo 1 $start "start $mode run"

    # ---- Wait for First Exposure ----
    expect {
        -re "exposure\[^\r\]*\[\r\n\]+" { }
        timeout { }
    }

    # ---- Wait for TX Trigger ----
    expect {
        -re "swHousekeeping\[^\r\]*\[\r\n\]+" {
            send -i $cmd_id "write 0 $txincraddr {\n $inc \n}\n"
            incr inc 500
            send -i $cmd_id "wait 1\nread 0 $txblock $txlen\n"
            exp_continue
        }
        -re "bepReadReply\[^\r\]*\
            commandId=1 \[^\r\]*\[\r\n\]+" {
            set state 1
        }
        -re "scienceReport\[^\r\]*\
            terminationCode=(\[0-9\]+\)\[\r\n\]+" {
            fail "BEP termination code $expect_out(1,string)"
        }
        -re "bepStartupMessage\[^\r\]*\[\r\n\]+" {
            fail "BEP rebooted in state $state"
        }
    }
    timeout {
        fail "Timeout: no RADMON bepReadReply state $state"
    }
}
```

```
# ---- Wait for bilevels to be reported ----
if {$state == 1} {
set timeout 60
set psci_id $spawn_id
spawn /bin/sh -c "filterClient -h $env(ACISSERVER) | ltlm -p61 -v"
expect {
    -re "value *= \[0-9\]+ \# \\(1011.*\[\r\n\]" { set state 2 }
    timeout { }
}
close
set spawn_id $psci_id
}

# ---- Stop the Job ----
send -i $cmd_id "stop 0 science\n"
wait_stop_science
system "ltlm -p1 -v pkts.raw | tail -34"

# ---- Report fate ----
if {$state == 2} {
send_user "\n---- RADMON triggered ----\n\n"
} else {
fail "RADMON not triggered $state"
}
}

# ---- Run a Raw Job ----
proc run_raw_test { mode rows start } {
    global cmd_id txblock txlen
    set timeout 360

    # ---- Load Image ----
    system make image$mode ROWS=$rows

    # ---- Load the TX parameter block ----
    load_tx_block

    # ---- Start the Science Run ----
    send -i $cmd_id "start 0 $mode 4\n"
    command_echo 1 $start "start raw $mode run"

    # ---- Wait for First Data Record ----
    expect {
        -re "data..Raw\[\r\]*\[\r\n\]+" { send -i $cmd_id "stop 0 science\n" }
        timeout { fail "No data packets" }
    }

    # ---- Dump the TX block ----
    send -i $cmd_id "read 2 $txblock $txlen\n"

    expect {
        -re "bepReadReply\[\r\]*\
            commandId=(\[0-9\])\[\r\]*\[\r\n\]+" {
            if {$expect_out(1,string) != 2} {
                fail "Bad bepReadReply id=$expect_out(1,string)"
            }
        }
        timeout { fail "No bepReadReply packet" }
    }

    # ---- Wait for End of Run ----
    set timeout 3600
    expect {
        -re "scienceReport\[\r\]*\

```

```
        terminationCode=(\[0-9]+\)\[\\r\\n]+" {
        if {$expect_out(1,string) != 1} {
        fail "BEP error $expect_out(1,string)"
        }
    }
    timeout {
        fail "Timed out after $timeout secs"
    }
}

# ---- Check that the TX block is unused ----
set cmd "ltlm -pl -v pkts.raw | tail -24 | diff - txings.txt"
if {[catch {system $cmd} err]} {
    fail $err
}
}

# ---- Run TE 3x3 Job ----
load_te_test 2 0 $rows $primary 580 665
run_event_test te $rows 15 14

# ---- Run TE EvHist Job ----
set ccd_list "7 0 1 2 10 5"
load_te_test 2 3 $rows 32 580 665
run_event_test te $rows 15 14

# ---- Run CC 3x3 Job ----
set ccd_list "7 0 1 2 3 5"
load_cc_test 2 0 285 330
run_event_test cc 512 17 16

# ---- Run CC 1x3 Job ----
load_cc_test 1 0 320 335
run_event_test cc 512 17 16

# ---- Run TE Raw Job ----
load_te_test 0 0 100 32 580 665
run_raw_test te 100 14

# ---- Run CC Raw Job ----
load_cc_test 0 0 285 330
run_raw_test cc 512 16

# ---- Success ----
pass "RADMON tests successful"
```

```
#!/usr/bin/env expect
#
# $Source: /nfs/acis/h3/acisfs/confignt1/patches/txings/testsuite/smoke/runtest2.tcl,v $
#
# Tests of txings patch
#

send_user "Welcome to txings/testsuite/smoke/runtest2.tcl\n"

# ---- Launch the command and telemetry server processes ----
set basedir [lindex $argv 0] ; # Patch base directory
set tools [lindex $argv 1] ; # Tool directory
set patchdir [lindex $argv 2] ; # Patch directory

# ---- Run Parameters ----
set ccd_list "7 0 1 2 3 5" ; # Desired fepCcdSelect
set rows 1024 ; # Number of rows in TE mode
set primary 32 ; # TE mode exposure time
set txminutes 3 ; # TX integration Time
set timeout 30 ; # Default timeout

# ---- Embed the Procedure Library ----
source $basedir/$tools/lib/lib-exp/runtest_support.tcl

# ---- Start the Command Pipe ----
spawn $basedir/$tools/bin/cmdclient $env(ACISSERVER)
set cmd_id $spawn_id

# ---- Start the Telemetry Pipe ----
spawn $basedir/$tools/bin/tlmclient $env(ACISSERVER)
sleep 1

# ---- Load TX Parameters ----
source "txings.def"

# ---- Halt the Processor ----
cold_boot

# ---- Load Patches ----
load_patch_list "$basedir/$tools/share/standard.bcml\
    $basedir/$tools/share/opt_smtimedlookup.bcml\
    $basedir/$tools/share/opt_eventhist.bcml\
    $basedir/$tools/share/opt_cc3x3.bcml\
    $basedir/$tools/share/opt_tlmio.bcml\
    $basedir/$tools/share/opt_printshouse.bcml\
    $basedir/$tools/share/opt_deaeng.bcml\
    $basedir/$patchdir/opt_txings.bcml\
    $basedir/$patchdir/opt_fepthrottle.bcml"

# ---- Apply Patches ----
warm_boot

# ---- Power Down all Boards ----
power_off_boards
set timeout 10
expect { timeout {} }

# ---- Wait for FEPs to finish powering ----
for {set fep 5} {$fep >= 0} {incr fep -1} {
    if {[lindex [split $ccd_list " "] $fep] < 10} { break }
}
set timeout 120
power_on_boards "$ccd_list"
expect {
```



```
-re ".*SWSTAT_FEP_EXECPMEM: $fep\[\r\n]" { }
timeout { fail "boards not powered on" }
}

# ---- Select FEP input from DEA ----
system make deaselect

# ---- Initialize the TX Blocks ----
proc load_tx_block {} {
    global cmd_id txnext txblock txlen txminutes

    # ---- Load the TXnext parameter block ----
    send -i $cmd_id "write 0 $txnext {\n $txminutes 3 512 145920 700 40 \n}\n"
    command_echo 1 192 "write TXnext block"

    # ---- Clear the TX block ----
    send -i $cmd_id "write 0 $txblock {\n"
    for {set i 0} {$i < $txlen} {incr i} {send -i $cmd_id "0\n" }
    send -i $cmd_id "}\n"
    command_echo 1 192 "clear TX counters"
}

# ---- Load TE Parameter Block ----
proc load_te_test { fepmode bepmode rows primary thr vid } {
    global cmd_id ccd_list

    set th1 [expr $thr + 1]
    set th2 [expr $thr + 2]
    set th3 [expr $thr + 3]

    set p05 [expr $vid + 5]
    set p10 [expr $vid + 10]
    set m05 [expr $vid - 5]
    set m10 [expr $vid - 10]

    # --- Send Parameter Block ---
    send -i $cmd_id "load 0 te 4 {
parameterBlockId      = 0xffffffff
fepCcdSelect          = $ccd_list
fepMode               = $fepmode
bepPackingMode        = $bepmode
onChip2x2Summing      = 0
ignoreBadPixelMap     = 1
ignoreBadColumnMap    = 1
recomputeBias         = 0
trickleBias           = 0
subarrayStartRow      = 0
subarrayRowCount      = [expr $rows - 1]
overclockPairsPerNode = 8
outputRegisterMode    = 0
ccdVideoResponse      =      0      0      0      0      0      0
primaryExposure       = $primary
secondaryExposure     = 0
dutyCycle             = 0
fep0EventThreshold    = $th3  $th3  $th3  $th3
fep1EventThreshold    = $th3  $th3  $th3  $th3
fep2EventThreshold    = $thr   $thr   $thr   $thr
fep3EventThreshold    = $thr   $thr   $thr   $thr
fep4EventThreshold    = $th2   $th2   $th2   $th2
fep5EventThreshold    = $th3   $th3   $th3   $th3
fep0SplitThreshold    = 13     13     13     13
fep1SplitThreshold    = 13     13     13     13
fep2SplitThreshold    = 13     13     13     13
fep3SplitThreshold    = 13     13     13     13
```

```
fep5SplitThreshold = 13 13 13 13
fep4SplitThreshold = 13 13 13 13
fep5SplitThreshold = 13 13 13 13
lowerEventAmplitude = 2000
eventAmplitudeRange = 10
gradeSelections = 0xfeffffff 0xffffffff 0xffffffffb 0xfffff7ff\
                  0xffffffff 0xffffffff 0xffbfffff 0x7fffffff

windowSlotIndex = 65535
histogramCount = 10000
biasCompressionSlotIndex = 1 1 1 1 1 1
rawCompressionSlotIndex = 0
ignoreInitialFrames = 2
biasAlgorithmId = 1 1 1 1 1 1
biasArg0 = 5 5 5 5 5 5
biasArg1 = 10 10 10 10 10 10
biasArg2 = 20 20 20 20 20 20
biasArg3 = 50 50 50 50 50 50
biasArg4 = 20 20 20 20 20 20
fep0VideoOffset = $vid $p05 $p05 $p05
fep1VideoOffset = $vid $vid $vid $vid
fep2VideoOffset = $p10 $vid $p10 $m10
fep3VideoOffset = $vid $vid $vid $vid
fep4VideoOffset = $vid $vid $vid $vid
fep5VideoOffset = $vid $vid $vid $vid
deaLoadOverride = 0
fepLoadOverride = 0\n}\n"

# ---- Wait for Echo ----
command_echo 1 9 "load te"
}

# ---- Load CC Parameter Block ----
proc load_cc_test { fepmode bepmode thr vid } {
    global cmd_id ccd_list

    set th1 [expr $thr + 1]
    set th2 [expr $thr + 2]
    set th3 [expr $thr + 3]

    set p05 [expr $vid + 5]
    set p10 [expr $vid + 10]
    set m05 [expr $vid - 5]
    set m10 [expr $vid - 10]

    # --- Send Parameter Block ---
    send -i $cmd_id "load 0 cc 4 {
parameterBlockId = 0xffffffff
fepCcdSelect = $ccd_list
fepMode = $fepmode
bepPackingMode = $bepmode
ignoreBadColumnMap = 1
recomputeBias = 0
trickleBias = 0
rowSum = 0
columnSum = 0
overclockPairsPerNode = 8
outputRegisterMode = 0
ccdVideoResponse = 0 0 0 0 0 0
fep0EventThreshold = $th3 $th3 $th3 $th3
fep1EventThreshold = $th3 $th3 $th3 $th3
fep2EventThreshold = $thr $thr $thr $thr
fep3EventThreshold = $thr $thr $thr $thr
fep4EventThreshold = $th2 $th2 $th2 $th2
fep5EventThreshold = $th3 $th3 $th3 $th3
```

```
fep0SplitThreshold      = 13 13 13 13
fep1SplitThreshold      = 13 13 13 13
fep2SplitThreshold      = 13 13 13 13
fep3SplitThreshold      = 13 13 13 13
fep5SplitThreshold      = 13 13 13 13
fep4SplitThreshold      = 13 13 13 13
fep5SplitThreshold      = 13 13 13 13
lowerEventAmplitude     = 2000
eventAmplitudeRange     = 10
gradeSelections         = 0x0e
windowSlotIndex         = 65535
rawCompressionSlotIndex = 0
ignoreInitialFrames     = 2
biasAlgorithmId         = 1 1 1 1 1 1
biasRejection           = 384 384 384 384 384 384
fep0VideoOffset         = $vid $p05 $p05 $p05
fep1VideoOffset         = $vid $vid $vid $vid
fep2VideoOffset         = $p10 $vid $p10 $m10
fep3VideoOffset         = $vid $vid $vid $vid
fep4VideoOffset         = $vid $vid $vid $vid
fep5VideoOffset         = $vid $vid $vid $vid
deaLoadOverride         = 0
fepLoadOverride         = 0\n}\n"

# ---- Wait for Echo ----
command_echo 1 10 "load cc"
}

# ---- Run Event Mode Test ----
proc run_event_test { mode rows bias start } {
    global cmd_id spawn_id env txblock txlen txincraddr
    set timeout 360
    set state 0
    set inc 0

    # ---- Load the TX parameter block ----
    load_tx_block

    # ---- Compute Bias Maps ----
    send -i $cmd_id "start 0 $mode bias 4\n"
    command_echo 1 $bias "start bias run"
    wait_stop_science

    # ---- Load the TX parameter block ----
    load_tx_block

    # ---- Start the Science Run ----
    send -i $cmd_id "start 0 $mode 4\n"
    command_echo 1 $start "start $mode run"

    # ---- Wait for First Exposure ----
    expect {
        -re "exposure\[^\r\]*\[\r\n\]+" { }
        timeout { }
    }

    # ---- Wait for TX Trigger ----
    expect {
        -re "swHousekeeping\[^\r\]*\[\r\n\]+" {
            send -i $cmd_id "write 0 $txincraddr {\n $inc \n}\n"
            incr inc 500
            send -i $cmd_id "wait 1\nread 0 $txblock $txlen\n"
            exp_continue
        }
    }
}
```

```
-re "bepReadReply\[^\r]*\  
  commandId=1 \[^\r]*\[r\n]+" {  
  set state 1  
}  
-re "scienceReport\[^\r]*\  
  terminationCode=(\[0-9+\)\[r\n]+" {  
  fail "BEP termination code $expect_out(1,string)"  
}  
-re "bepStartupMessage\[^\r]*\[r\n]+" {  
  fail "BEP rebooted in state $state"  
}  
timeout {  
  fail "Timeout: no RADMON bepReadReply state $state"  
}  
}  
  
# ---- Wait for bilevels to be reported ----  
if {$state == 1} {  
  set timeout 60  
  set psci_id $spawn_id  
  spawn /bin/sh -c "filterClient -h $env(ACISSERVER) | ltlm -p61 -v"  
  expect {  
    -re "value *= \[0-9]+ \# \\\(1011.*\[r\n]*" { set state 2 }  
    timeout { }  
  }  
  close  
  set spawn_id $psci_id  
}  
  
# ---- Stop the Job ----  
send -i $cmd_id "stop 0 science\  
set timeout 600  
expect {  
-re "scienceReport\[^\r]*\  
  terminationCode=(\[0-9+\)\[r\n]+" {  
  if {$expect_out(1,string) != 1} {  
    fail "BEP termination code $expect_out(1,string)"  
  }  
}  
timeout {  
  fail "Timeout: no scienceReport in state $state"  
}  
}  
  
# ---- Report triggering TXblock ----  
system "ltlm -p1 -v pkts.raw | tail -33"  
  
# ---- Report fate ----  
if {$state == 2} {  
  send_user "\n---- RADMON triggered ----\  
} else {  
  fail "RADMON not triggered $state"  
}  
}  
  
# ---- Run a Raw Job ----  
proc run_raw_test { mode rows start } {  
  global cmd_id txblock txlen  
  set timeout 360  
  
  # ---- Load the TX parameter block ----  
  load_tx_block  
  
  # ---- Start the Science Run ----
```

```
send -i $cmd_id "start 0 $mode 4\n"
command_echo 1 $start "start raw $mode run"

# ---- Wait for First Data Record ----
expect {
-re "data..Raw\[^\r\]*\[\\r\\n]+" {
    send -i $cmd_id "stop 0 science\n"
}
timeout {
    fail "No data packets"
}
}

# ---- Dump the TX block ----
send -i $cmd_id "read 2 $txblock $txlen\n"
expect {
-re "bepReadReply\[^\r\]*\
    commandId=(\[0-9\])\[^\r\]*\[\\r\\n]+" {
    if {$expect_out(1,string) != 2} {
        fail "Bad bepReadReply id=$expect_out(1,string)"
    }
}
timeout {
    fail "No bepReadReply packet"
}
}

# ---- Wait for End of Run ----
set timeout 3600
expect {
-re "scienceReport\[^\r\]*\
    terminationCode=(\[0-9+\])\[\\r\\n]+" {
    if {$expect_out(1,string) != 1} {
        fail "BEP termination code $expect_out(1,string)"
    }
}
timeout {
    fail "Timeout: no scienceReport in state $state"
}
}

# ---- Check that the TX block is unused ----
set cmd "ltlm -p1 -v pkts.raw | tail -23 | diff - txings.txt"
if {[catch {system $cmd} err]} {
    fail $err
}
}

# ---- Run TE 3x3 Job ----
load_te_test 2 0 $rows $primary 4 33
run_event_test te $rows 15 14

# ---- Run TE EvHist Job ----
set ccd_list "7 0 1 2 10 5"
load_te_test 2 3 $rows 32 4 33
run_event_test te $rows 15 14

# ---- Run CC 3x3 Job ----
set ccd_list "7 0 1 2 3 5"
load_cc_test 2 0 4 33
run_event_test cc 512 17 16

# ---- Run CC 1x3 Job ----
load_cc_test 1 0 4 33
```

```
run_event_test cc 512 17 16

# ---- Run TE Raw Job ----
load_te_test 0 0 100 32 4 33
run_raw_test te 100 14

# ---- Run CC Raw Job ----
load_cc_test 0 0 4 33
run_raw_test cc 512 16

# ---- Success ----
pass "RADMON tests successful"
```

```
#!/usr/bin/perl
#
# $Source: /nfs/acis/h3/acisfs/confignt1/patches/txings/testsuite/smoke/makebias.pl,v $
#

$rows = $ARGV[0] ? $ARGV[0] : 1024;
$noop = $ARGV[1] ne '' ? $ARGV[1] : 33743;

print <<EOF;
  Rows      = $rows
  Columns   = 256
  Mode      = ABCD
  Overclocks = 16
  Seed      = 12345678
  Noop      = 4 before Oclks
  Noop      = 0 before HSYNC
  Noop      = 8 after  HSYNC
  Noop      = $noop before VSYNC
  Noop      = 3 after  VSYNC

  Begin Node = A
    Bias     = 210
    dBias    = 0
    OverClock = 200
    dOverClock = 0
  End Node   = A

  Begin Node = B
    Bias     = 310
    dBias    = 0
    OverClock = 300
    dOverClock = 0
  End Node   = B

  Begin Node = C
    Bias     = 410
    dBias    = 0
    OverClock = 400
    dOverClock = 0
  End Node   = C

  Begin Node = D
    Bias     = 510
    dBias    = 0
    OverClock = 500
    dOverClock = 0
  End Node   = D
EOF

exit 0;
```

```
#!/usr/bin/perl
#
# $Source: /nfs/acis/h3/acisfs/confignt1/patches/txings/testsuite/smoke/makeimage.pl,v $
#

$rows = $ARGV[0] ? $ARGV[0] : 1024;
$noop = $ARGV[1] ne '' ? $ARGV[1] : 34626;
$incr = $ARGV[2] ? $ARGV[2] : 100;

print <<EOF;
  Rows      = $rows
  Columns   = 256
  Mode      = ABCD
  Overclocks = 16
  Seed      = 12345678
  Noop      = 4 before Oclks
  Noop      = 0 before HSYNC
  Noop      = 8 after  HSYNC
  Noop      = $noop before VSYNC
  Noop      = 3 after  VSYNC

  Begin Node = A
    Bias     = 220
    dBias    = 0
    OverClock = 201
    dOverClock = 0
  End Node   = A

  Begin Node = B
    Bias     = 320
    dBias    = 0
    OverClock = 302
    dOverClock = 0
  End Node   = B

  Begin Node = C
    Bias     = 420
    dBias    = 0
    OverClock = 403
    dOverClock = 0
  End Node   = C

  Begin Node = D
    Bias     = 520
    dBias    = 0
    OverClock = 504
    dOverClock = 0
  End Node   = D
EOF

$r2 = 5;
$c2 = 5;
$rr = 2 * $r2 + 1;
$cc = 2 * $c2 + 1;
$n = 1;

# for ($r = $rr; $r < $rows - $rr - 1; $r += $incr) {

for ($r = $rows - $rr - 1; $r > $rr; $r -= $incr) {
  for ($c = $cc; $c < 1024 - $cc; $c += $incr, $n++) {
    $v = " $n" x ($rr * $cc);
    print <<EOE;
    Begin Event = event_$$n
      Rows      = $rr
```



```
Columns = $cc  
Value = $v  
End Event = event_$n  
  
event_$n $r $c  
EOE  
}  
}  
  
exit 0;
```

TITLE: ACIS Flight Software Standard Patch Component Release Notes

DOCUMENT NUMBER: 36-58010 REVISION: E

ORIGINATOR: Peter G. Ford <pgf@space.mit.edu>

LETTER	SCO NO.	DESCRIPTION	APPROVED	DATE
01	36-984	Initial numeric release	jimf	10/27/1998
A	36-1006	Bug fixes, incorporate tests	RFG	05/11/1999
B	36-1019	Add new patches, retest	RFG	12/16/1999
C	36-1035	Add new patches, retest	RFG	08/09/2007
D	36-1039	Add new patches, retest	RFG	09/29/2009
E	36-1042	Update buscrash2, retest	RFG	01/06/2010
E	36-1042	Update buscrash2		

=====
Title: ACIS Patch Release Notes for Version E

Software Change Order: 36-1042

Build Date: Sun May 22 11:08:30 EDT 2011
Part Number: 36-58010
Version: E
CVS Tag: release-E

IPCL Number: 36-53204.0204
IPCL Version: N
IPCL CVS Tag: release-N

Load Size: 2660 bytes

Description:

This is the fifth letter release of the standard patch set for the ACIS Flight Software.

The purpose of this release is to add the txings optional patch along with the fepthrottle patch that is used to test it..

This release consists of the following bug fix/system modification patches, where * indicates the new or modified patches since the previous release:

biastiming	- Fixes	SPR 117
corruptblock	- Fixes	SPR 113
digestbiaserror	- Fixes	SPR 116
histogramvar	- Fixes	SPR 115
rquad	- Fixes	SPR 121
histogrammean	- Fixes	SPR 123
zaplexpo	- Addresses	SPR 122
condoclk	- Addresses	SPR 127
fepbiasparity2	- Addresses	SPR 130
cornermean	- Fixes	SPR 128
tlmbusy	- Fixes	SPR 138
buscrash	- Fixes	SPR 140
badpix	- Fixes	SPR 141
buscrash2	- Fixes	SPR 142

For archival purposes, this document contains two attachments. The first contains ASCII command inputs to the ACIS command generator, "bcmd", used to generate the binary patch commands corresponding to this release. The second attachment contains the linker map listing for the ACIS Flight Software, and the patches built by this release.

The following documentation identifies these patches, provides a brief justification for each patch, and briefly describes the contents of these patches and their command, telemetry and science impacts.

Addressed Problem Reports:

SPR-142
SPR-128
SPR-123
SPR-127
SPR-130
SPR-138

SPR-122
SPR-141
SPR-115
SPR-113
SPR-140
SPR-117
SPR-116
SPR-121

Included Patches:

tlmbusy
fepbiasparity2
biastiming
histogramvar
badpix
zaplexpo
digestbiaserror
corruptblock
cornermean
buscrash
buscrash2
rquad
condock
histogrammean

Additional Release Level Tests:

=====
Patch Name: tlmbusy

Part Number: 36-58030.29

Version: A

SCO:

Description:

This standard patch prevents the BEP from writing anomalous telemetry output when the TlmManager::post() method is called from one task while it is still enqueueing a packet from another task.

The BEP will not drop the occasional packet (usually a housekeeping packet), and will be prevented from writing garbage in its stead. This will prevent the ground system from mis-processing science runs in which the garbage consists of correctly formatted, but unexpected, packets.

Applicable Reports/Requests:

SPR-138
SER-None

Test Results:

smoke --> PASS

Replaced Functions:

TlmManager::post

Command Impact:

None.

Telemetry Impact:

The occasional packet drop-out or garbling will no longer occur, so the impact should be wholly favorable.

Science Impact:

None.

=====
Patch Name: fepbiasparity2

Part Number: 36-58030.19
Version: A
SCO: 36-1015

Description:

In TE mode, this patch causes FEP_0 to bypass the upper half of each image map (rows 512 through 1023) if the bias parity errors in any one frame reported by the firmware exceed a threshold value (10). In addition, the 10 bias values, and their corresponding pixel values, are copied to a static location from which they can be dumped at a later time. In CC mode, the patch copies the lower half of the FEP_0 bias map into the upper half whenever 10 or more bias errors have been detected.

The patch has no effect on other FEPs.

Applicable Reports/Requests:

SPR-130

Test Results:

bugTe --> PASS
bugCc --> PASS
fixTe --> PASS
patchCc --> PASS

Replaced Functions:

Command Impact:

Once the patch is installed and FEP_0 powered up and running, it is advisable to clear its static save area via the following command:

```
write 'c' fep 0 0x80000210 {  
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
}
```

Then, either on a regular basis, or when it is noticed that 10 parity errors have been reported from a single FEP_0 exposure frame, the following command should be executed to dump the contents of the static save area:

```
read 'c' fep 0 0x80000210 20
```

Telemetry Impact:

If 10 or more bias parity errors are detected in FEP_0 during a timed-exposure science run, fepbiasparity2 will prevent more from being reported in telemetry. Once the threshold is reached, no further events will be reported from rows 512-1023. In 5x5 mode, a few additional parity errors may be reported from row 512.

In continuous clocking mode, when 10 or more bias parity errors are detected in FEP_0, fepbiasparity2 will copy the entire contents of the lower half of the bias map, i.e., 512 rows x 1024 pixels, to the upper half, thereby (hopefully) restoring the original contents. Occasional

parity errors will be corrected in the usual manner, i.e., by searching through the bias map, starting at row 0, for a pair of undamaged values.

Science Impact:

When this patch is triggered in timed-exposure modes, no further parity errors will be reported from rows 513-1023 of the CCD attached to FEP_0. In 3x3 mode, no events will be reported from rows 511-1023; in 5x5 mode, none will be reported from 510-1023. Ground software must be prepared to sense this condition, e.g., by examining the biasParityErrors fields in exposure packets, or by recognizing the absence of events above row 512, and updating the exposure maps accordingly.

The patch should have less impact in continuous clocking mode. When the 10-error threshold is triggered, FEP_0 may skip an exposure frame while replacing the upper half of its bias map, but otherwise, event processing will continue, taking advantage of the full area of the CCD.

=====
Patch Name: biastiming

Part Number: 36-58030.04

Version: A

SCO: 36-993

Description:

Reason:

This patch fixes a software problem which was first encountered during AXAF thermal vacuum testing at TRW.

Symptom:

At TRW thermal vacuum testing, someone observed that the instrument sent a science report in the middle of trickled bias map data. Bev has subsequently observed one case where the instrument started sending science data while trickling the bias maps.

Symptom Impact:

This symptom opens the possibility that the FEP threshold plane will lock up during a science run if the event rate is high enough (on the order of 5K events/sec/CCD).

Symptom Cause:

When the science manager tells the bias thief to start, by calling biasReady(), it set the thief's busy flag prior to signaling the task to start. If the task monitor sneaks in, the bias thief's main loop, goTaskEntry() ends up re-clearing the busyFlag, but then later picks up the start event and starts trickling the bias map. Since the busyFlag is clear at this point, the science manager assumes that the bias has been sent, and proceeds on to the data processing portion of the run (or if it's a bias only run or the run has been told to stop, the terminate the run).

Fix Description:

This patch replaces the BiasThief::biasReady() function with one that re-orders the setting of the busyFlag. In the patched version, the busyFlag is set AFTER the notification to the thief to start sending the bias. If the task monitor sneaks in, the thief will clear the flag, but once we return to the biasReady() function, the flag will be correctly asserted.

Applicable Reports/Requests:

SPR-117

Test Results:

unit --> PASS

fix --> PASS

Replaced Functions:

BiasThief::biasReady

Command Impact:

None

Telemetry Impact:

When this patch is not installed, it is possible, but rare, for bias maps to be telemetered while data processing is running and telemetering event data and exposure records, and even for a science report to be issued while the bias maps continue to be telemetered.

Once the patch is installed, the instrument will reliably wait until all of the bias maps have been telemetered before proceeding with the data processing portion of the run.

Science Impact:

Without this patch, it is possible, but extremely unlikely, that the FEP hardware threshold plane may lockup. This results in unreasonably low energy events being reported in the same set of positions, where ever there was a threshold crossing at the point where the threshold hardware locked up. This occurrence has only been seen with high event rates, on the order of 3000-5000 per exposure.

With this patch, this situation will not occur.

=====
Patch Name: histogramvar

Part Number: 36-58030.03

Version: A

SCO: 36-999

Description:

This patch fixes a software problem, SPR-115.

Symptom:

The Raw Histogram Mode occassionally produces anomalously large values for the low word of the overclock variances.

Symptom Impact:

This slightly degrades the science analysis of histogram mode data by very occassionally providing bad variance values for the overlocks.

Symptom Cause:

The error is cause by an unsigned integer divide which should have been a signed integer divide. If the low order word ends up negative this produces an incorrectly high value for the variance.

Fix Description:

This inline patch modifies the FEP to use a signed divide instead of unsigned divide.

Applicable Reports/Requests:

SPR-115

Test Results:

reproduce --> PASS

fix --> PASS

Replaced Functions:

Command Impact:

None

Telemetry Impact:

None

Science Impact:

This patch affects Histogram Mode Only.

Without this patch, the overclock variances in histogram mode may occassionally be incorrect. Once this patch is installed, the Flight Software correctly computes overclock variances.

=====
Patch Name: badpix

Part Number: 36-58030.21

Version: A

SCO: 36-1037

Description:

Reason:

This patch fixes software problem report SPR-141.

Symptom:

The known bad pixels and columns supplied to ACIS through its bad pixel and column lists are not always being flagged in the correct locations in the FEP bias maps. The symptom only appears when the instrument is running in timed-exposure mode using sub-arrays whose initial row number is greater than zero.

Symptom Impact:

In most timed-exposure sub-array runs, when the sub-array starts after the first CCD row, bad pixel will be mis-located; the truly bad pixels will be accepted as valid and good pixels will be treated as bad. In practice, this will have little effect since bad pixels will be recognized by the bias map creation algorithm.

Symptom Cause:

The BEP maintains a list of known bad pixels and columns in each CCD. After a bias map is created, the BEP's loadBadMaps procedure will set the appropriate entries in the FEPs bias maps to 4095, telling the FEP software to ignore the corresponding image pixel, i.e., treat it as if it had zero value. This is in addition to any saturated pixels found during bias map creation, which will also be assigned the bias value 4095.

The code in SmTimedExposure::loadBadMaps() contains an error. It assumes that sub-arrays will be processed in the same relative location in a FEP's image and bias memory as on the CCD from which the pixels originated. This is not so--the first row of a sub-array is always written into row 0 of a FEP's image map, and the corresponding bias values are saved in row 0 of its bias map.

SmTimedExposure::loadBadMaps() must be patched in two places, one to correct bad pixels, the other bad columns. The bad pixel correction is applied as follows:

```
while (badPixelMap.getPixel (index, ccd, row, col) == BoolTrue) {
  if ((row >= start) && (row < end)) {
    row /= sum;
    col /= sum;
    for (FepId fep = FEP_0; fep < FEP_COUNT; fep = FepId(fep+1)) {
      if (fepCcd[fep] == ccd) {
        fepManager.loadBadPixel (fep, row, col);
      }
    }
  }
  index++;
}
```

and we want to change the "row /= sum" to "row = (row-start) / sum". This can best be done by recognizing that "sum" has only two values, 1 or 2, and the MIPS takes 32 bytes of code to perform an unsigned integer divide, but only 4 bytes to perform a logical right shift.

The original assembler code

```
1774 2400A28F   lw      $2,36($sp)
1778 00000000   divu    $2,$2,$18
177C 1B005200
1780 02004016
1784 00000000
1788 0D000700
1798 2400A2AF   sw      $2,36($sp)
```

can simply be modified as follows:

```
1774 2400A28F   lw      $2,36($sp)
1778 FFFF4326   addu    $3,$18,-1
177c 23105600   subu    $2,$2,$22
1780 06106200   srl     $2,$2,$3
1784 00000000   nop
1788 00000000   nop
178C 00000000   nop
1790 00000000   nop
1794 00000000   nop
1798 2400A2AF   sw      $2,36($sp)
```

The second patch sets the starting value of the row loop to zero:

```
while (badTeColumnMap.getColumn (index, ccd, col) == BoolTrue) {
    col /= sum;
    for (FepId fep = FEP_0; fep < FEP_COUNT; fep = FepId(fep+1)) {
        if (fepCcd[fep] == ccd) {
            for (unsigned row = start; row < end; row++) {
                fepManager.loadBadPixel (fep, row, col);
            }
        }
    }
    index++;
}
```

The existing assembler code is

```
$LM1578:
18cc 0000043C   la     $4,fepManager
18d0 00008424
18d4 21282002   move   $5,$17
18d8 3000A78F   lw     $7,48($sp)
18dc 00000000   nop
18e0 0000000C   jal    loadBadPixel
18e4 21300002   move   $6,$16
18e8 01001026   addu   $16,$16,1
18ec 2B101402   sltu   $2,$16,$20
18f0 F6FF4014   bne    $2,$0,$L1578
```

and the patch replaces the row in the loadBadPixel(fepId, row, col) call with row-start. (In the MIPS architecture, the instruction after a branch or call is executed before the branch is taken).

```
18e4 23301602   subu   $6,$16,$22
```

Applicable Reports/Requests:
SPR-141

Test Results:

reproduce --> PASS
fix --> PASS

Replaced Functions:

Command Impact:
None.

Telemetry Impact:
None.

Science Impact:
Without this patch, the BEP's bad pixel and bad column lists will be applied incorrectly in timed-exposure sub-array mode when the sub-array begins on any but the first row of the CCD. Since almost all science runs are made in dithered mode, the impact once the patch is in place will be slight.

=====
Patch Name: zaplexpo

Part Number: 36-58030.16

Version: A

SCO: 36-997

Description:

Reason:

In event-finding mode, the FEP thresholds are adjusted using delta-overclock values, which are calculated from difference between the average overclock values from the preceding frame and the average overclock values from the initial bias frame. The delta-overclocks for the initial data frame are set to zero, i.e., it is assumed that the mean bias levels haven't drifted since the first exposure frame used to compute the bias map. This is often a poor assumption, and can lead to a very large number of events being reported within the first exposure.

Fix Description:

Inhibit the FEP from finding any threshold crossings within the first examined exposure frame. This is performed at science run initialization time within the "fepSciTimed.c":FEPsciTimedInit function (TE mode) and the "fepSciCclk.c":FEPsciCclkInit function (CC mode) by storing 4095 in the FEP threshold registers. Thus,

```

186:fepSciTimed.c ****   for (iquad = 0; iquad < 4; iquad++) {
925 0290 21200000           move    $4,$0
926 0294 0000053C           la      $5,stageThresh
926      0000A524
187:fepSciTimed.c ****   fp->ex.bias0[iquad] = fp->br.bias0[iquad];
929 029c 40100400           sll    $2,$4,1
930                                $L90:
931 02a0 21105000           addu   $2,$2,$16
932 02a4 A0024394           lhu    $3,672($2)
933 02a8 00000000
934 02ac 100043A4           sh     $3,16($2)
188:fepSciTimed.c ****   fp->ex.dOclk[iquad] = 0;
937 02b0 180040A4           sh     $0,24($2)
189:fepSciTimed.c ****   FIOsetThresholdRegister(iquad, (short)(fp->tp.thresh[iqu
ad]));
944 02b4 80180400           sll    $3,$4,2
945 02b8 21107000           addu   $2,$3,$16
948 02bc 21186500           addu   $3,$3,$5
949 02c0 4C004284           lh     $2,76($2)
950 02c4 00000000
951 02c8 000062AC           sw     $2,0($3)
958 02cc 01008424           addu   $4,$4,1
959 02d0 0400822C           sltu   $2,$4,4
960                                .set   noreorder
961                                .set   nomacro
962 02d4 F2FF4014           bne    $2,$0,$L90
963 02d8 40100400           sll    $2,$4,1
964                                .set   macro
965                                .set   reorder
190:fepSciTimed.c ****   }

```

becomes

```

186:fepSciTimed.c ****   for (iquad = 0; iquad < 4; iquad++) {
925 0290 21200000           move    $4,$0
926 0294 0000053C           la      $5,stageThresh
926      0000A524

```

```

187:fepSciTimed.c ****      fp->ex.bias0[iquad] = fp->br.bias0[iquad];
929 029c 40100400          sll      $2,$4,1
930                          $L90:
931 02a0 21105000          addu    $2,$2,$16
932 02a4 A0024394          lhu     $3,672($2)
933 02a8 00000000
934 02ac 100043A4          sh      $3,16($2)
188:fepSciTimed.c ****      fp->ex.dOclk[iquad] = 0xffff;
937 02b0 FF0F0324          li      $3,0x00000fff
944 02b4 180043A4          sh      $3,24($2)
189:fepSciTimed.c ****      FIOsetThresholdRegister(iquad, 0xffff);
945 02b8 80180400          sll    $3,$4,2
948 02bc 21186500          addu   $3,$3,$5
949 02c0 FF0F0224          li     $2,0x00000fff
950 02c4 00000000
951 02c8 000062AC          sw     $2,0($3)
958 02cc 01008424          addu   $4,$4,1
959 02d0 0400822C          sltu   $2,$4,4
960                          .set   noreorder
961                          .set   nomacro
962 02d4 F2FF4014          bne    $2,$0,$L90
963 02d8 40100400          sll    $2,$4,1
964                          .set   macro
965                          .set   reorder
190:fepSciTimed.c ****      }

```

and

```

174:fepSciCCLK.c ****      for (iquad = 0; iquad < 4; iquad++) {
774 01fc 21200000          move   $4,$0
775 0200 0000053C          la     $5,stageThresh
775      0000A524
175:fepSciCCLK.c ****      fp->ex.bias0[iquad] = fp->br.bias0[iquad];
778 0208 40100400          sll    $2,$4,1
779                          $L83:
780 020c 21105000          addu   $2,$2,$16
781 0210 A0024394          lhu    $3,672($2)
782 0214 00000000
783 0218 100043A4          sh     $3,16($2)
176:fepSciCCLK.c ****      fp->ex.dOclk[iquad] = 0;
786 021c 180040A4          sh     $0,24($2)
177:fepSciCCLK.c ****      FIOsetThresholdRegister(iquad, (short)(fp->tp.thresh[iqu
ad]));
793 0220 80180400          sll    $3,$4,2
794 0224 21107000          addu   $2,$3,$16
797 0228 21186500          addu   $3,$3,$5
798 022c 4C004284          lh     $2,76($2)
799 0230 00000000
800 0234 000062AC          sw     $2,0($3)
807 0238 01008424          addu   $4,$4,1
808 023c 0400822C          sltu   $2,$4,4
809                          .set   noreorder
810                          .set   nomacro
811 0240 F2FF4014          bne    $2,$0,$L83
812 0244 40100400          sll    $2,$4,1
813                          .set   macro
814                          .set   reorder
178:fepSciCCLK.c ****      }

```

becomes

```

174:fepSciCCLK.c ****      for (iquad = 0; iquad < 4; iquad++) {
774 01fc 21200000          move   $4,$0
775 0200 0000053C          la     $5,stageThresh

```

```
775          0000A524
175:fepSciCclk.c ****      fp->ex.bias0[iquad] = fp->br.bias0[iquad];
778 0208 40100400          sll      $2,$4,1
779                                $L83:
780 020c 21105000          addu    $2,$2,$16
781 0210 A0024394          lhu     $3,672($2)
782 0214 00000000
783 0218 100043A4          sh      $3,16($2)
176:fepSciCclk.c ****      fp->ex.dOclk[iquad] = 0xffff;
786 021c FF0F0324          li      $3,0x00000fff
787 0220 180043A4          sh      $3,24($2)
177:fepSciCclk.c ****      FIOsetThresholdRegister(iquad, 0xffff);
793 0224 80180400          sll     $3,$4,2
797 0228 21186500          addu    $3,$3,$5
798 022c FF0F0224          li      $2,0x00000fff
799 0230 00000000
800 0234 000062AC          sw      $2,0($3)
807 0238 01008424          addu    $4,$4,1
808 023c 0400822C          sltu   $2,$4,4
809                                .set   noreorder
810                                .set   nomacro
811 0240 F2FF4014          bne     $2,$0,$L83
812 0244 40100400          sll     $2,$4,1
813                                .set   macro
814                                .set   reorder
178:fepSciCclk.c ****      }
```

Applicable Reports/Requests:
SPR-122

Test Results:
reproduce --> PASS
fix --> PASS

Replaced Functions:

Command Impact:
None

Telemetry Impact:
No events will be generated for the first examined exposure, i.e., the frame with exposureNumber == 2 (unless the teignore or ccignore patches are loaded, in which case it will be the frame with exposureNumber == ignoreInitialFrames).

To determine whether this patch was in effect during a particular science run, telemetry processing software should examine the 4 values in the deltaOverclocks array in exposure packets with exposureNumber == 2 (or with exposureNumber == ignoreInitialFrames if the relevant teignore or ccignore patch is installed). If they are all equal to 4095, the patch was installed and this exposure frame should not be included in the good time interval (GTI); if they are all zero, the patch was omitted.

Science Impact:
With this patch installed, the frame with exposureNumber == 2 (or with exposureNumber == ignoreInitialFrames if the relevant teignore or

ccignore patch is installed) should not be included in the GTI maps.

=====
Patch Name: digestbiaserror

Part Number: 36-58030.02
Version: A
SCO: 36-995

Description:

This patch fixes software problem SPR-116.

Symptom:

When a parity error is detected, the FEP produces a pair of bias values with a flag indicating if one or both are corrupt. The BEP mishandles this when telemetering the error. If the error occurs at an odd column position, the BEP reports the wrong column position of the error.

Symptom Impact:

This has the potential to degrade the science analysis by providing ambiguous knowledge of which bias map values have been corrupted.

Symptom Cause:

In PmEvent::digestBiasError, it assumes that only one of pair of bias values is corrupt and that the FEP reported column indicates which of the two is corrupt. This is WRONG.

Fix Description:

This inline patch provides a new representation of the bias error event and modifies the telemetry format tag to indicate the new format. Rather than telemeter the corrupt value (which is fairly useless), the 12-bit value field is as follows, where bit 0 is the least-significant bit:

Bits 0 - 3: The top 4 bits of the bias value at the column position
Bits 4 - 7: The top 4 bits of the bias value at column + 1
Bits 8 - 11: Unused

These bits contain the results of the hardware parity check of the corresponding pixel bias value.

The format of these 4 bits are as follows:

Bit 0 (H/W bit 12) - Always zero
Bit 1 (H/W bit 13) - H/W computed parity of bias map value
Bit 2 (H/W bit 14) - Parity bit stored in parity plane
Bit 3 (H/W bit 15) - Parity error bit (0 - no parity error, 1 - parity error)

The bit definition information is derived from the "DPA Hardware Specification and System Description", MIT 36-02104 Rev. C., Section 2.2.2.5.5 "Bias Map Parity Detection".

Applicable Reports/Requests:

SPR-116

Test Results:

reproduce --> PASS
fix --> PASS

Replaced Functions:

Command Impact:
None

Telemetry Impact:

This patch affects the telemetry Pixel Bias Map Error records. Without this patch, the error records will be incorrect if the error occurs on an odd column. With this patch installed, the instrument will telemetry bias errors using a new telemetry format, TTAG_SCI_PATCHED_BIAS_ERROR, defined by the "Patch Data Bias Error" format in the IP&CL Software Structures Definitions, MIT 36-53204.0204 Rev. L.

Science Impact:

Without the patch installed, there is an ambiguity whether a bias error is in the reported pixel, or in the adjacent, odd column. Once the patch is installed, the ground can determine exactly which pixel was upset.

=====
Patch Name: corruptblock

Part Number: 36-58030.01

Version: A

SCO: 36-994

Description:

Reason:

This patch fixes software problem report SPR-113.

Symptom:

If a parameter block is corrupt, the flight software may use nonsense parameters, if just powered on, or run the previous run mode's parameter block.

Symptom Impact:

If the original parameter block was corrupt and if this was the first run since the instrument was powered, the nonsense parameters may cause the instrument to crash and reset, preventing any science activity during that observation's time period. The system will recover, although without patches, at the onset of the next observation. If there was an earlier run of the same type, Timed Exposure or Continuous Clocking, the previous run's parameter will be used, which may or may not be ideal.

Symptom Cause:

The flight software start run routine, ChStartSciRun::processCmd(), declares an "alternate" parameter block variable, which is filled in by the science mode's checkBlock() routine if the original parameter block is corrupt. processCmd() then erroneously passes this "alternate", and a reference to the "alternate" back to checkBlock() to verify that the alternate is not also corrupt. The called checkBlock() initializes the 2nd reference to INVALID, which ends up overwriting the desired alternate block id. This propagates through to the run, preventing the mode from loading the parameter block, and using, instead, what it had already staged from an earlier run.

Fix Description:

This inline patch modifies 2nd parameter to refer to a dummy variable when checking the default backup block. This prevents the id from being overridden and provides the proper default parameter block selection behavior when the selected block has been corrupted.

The original line from chstartscirun.C is:

```
    if (mode.checkBlock (blockid, alternate) == BoolTrue)
    {
        result = CMDRESULT_OK;
    }
<<< else if (mode.checkBlock (alternate, alternate) == BoolTrue)
    {
        blockid = alternate;
        usedAlternate = BoolTrue;
    }
    else
    {
        return CMDRESULT_CORRUPT_IDLE;
    }
```

The effect of the patch changes this to:

```
if (mode.checkBlock (blockid, alternate) == BoolTrue)
{
    result = CMDRESULT_OK;
}
>>> else if (mode.checkBlock (alternate, dummy) == BoolTrue)
{
    blockid = alternate;
    usedAlternate = BoolTrue;
}
else
{
    return CMDRESULT_CORRUPT_IDLE;
}
```

The stack frame of the modified patch will appear as follows, where the offsets in the left-hand column are relative to the stack pointer at the time the jump is made to the called subroutine mode.checkBlock(), the symbols in the center column indicate the "conventional" locations for various registers, and the right column indicates if the assembler actually put anything into that stack slot. If "unassigned" then the assembler didn't explicitly store anything into that stack slot. If blank, then the "convention" (NOTE: In the MIPS processors, calls don't explicitly push anything on the stack. The return address is maintained in "ra" at the time of the call and the caller is then required to save it if needed):

```
*
* ChStartSciRun::processCmd() - Stack Frame
* Convention described in Section 2.3 of
* MIPS programmers handbook, by Farquahar and Bunce
*
* 60  pad      unassigned
* 56  ra       ra ($31)
* 52  s3       s3 ($19)
* 48  s2       s2 ($18)
* 44  s1       s1 ($17)
* 40  s0       s0 ($16)
* 36  f23      unassigned      (patch uses as local "dummy")
* 32  f22      alternate      (local variable)
* 28  f21      unassigned
* 24  f20      unassigned
* 20  pad      unassigned
* 16  arg      biasonly argument (arg4) to scienceManager.startRun()
* 12  a3       unassigned
* 8   a2       unassigned
* 4   a1       unassigned
* 0   a0       unassigned
```

Applicable Reports/Requests:
SPR-113

Test Results:
reproduce --> PASS
fix --> PASS

Replaced Functions:

Command Impact:
Without this patch, corruptions (if any are actually ever encountered) may cause an previous parameter block to be used for an observation, or

at worst, a reset of the instrument.

When the patch is installed, the instrument will use the appropriate default parameter block (slot 0 or slot 1) instead of the corrupted parameter block, or will skip the observation if the defaults are also corrupt.

Telemetry Impact:

None.

Although, without this patch, the instrument may select an inappropriate parameter block, the parameter blocks dumped to telemetry at the start of a science run will always be the the ones actually used for the run.

Science Impact:

None

=====
Patch Name: cornermean

Part Number: 36-58030.21

Version: A

SCO: 36-1017

Description:

Reason:

This patch fixes software problem report SPR-128.

Symptom:

In Timed Exposure Graded Telemetry mode, when some of the corner pixels have a small negative corrected pulse height, the system reports an incorrect, extremely large negative value for the mean corrected pulse height of the corner pixels. Additionally, the algorithm rounds incorrectly when the mean pulse height is negative (not mentioned in the SPR).

Symptom Impact:

Barring corrective ground analysis and action, the incorrectly reported corner mean value may confuse the science analysis process, and at worst, lead to incorrect conclusions about the science, or the state of the instrument data processing.

Symptom Cause:

The flight software routine, Pixel3x3:computePhGrade() divides a signed integer value, cornersum, with an unsigned integer value, sumcount (see filesscience/pixel3x3.H). In "C" and "C++", this division is performed as an unsigned divide, preventing any sign extension, hence the "signedness" of the cornersum is lost. The result is stored into a signed value, cornermean, which is later converted to a signed 13-bit value for telemetry. When the ground software extracts the 13-bit signed value, it will sign-extend the value. The effect of losing the sign in the divide, sometimes yields incorrect results, some of which appear as large negative values when processed by the ground.

The rounding problem is due to incorrect coding of the integer rounding for negative values:

```
mean = (sum + (count/2))/count
```

should be:

```
mean = (sum + (sign(sum) * int(count)/2))/int(count)
```

Fix Description:

This patch implements the fix to the loss of "signedness" problem and the rounding using an inline assembler patch.

To fix the loss of "signedness" problem the patch replaces the existing unsigned divide instruction (divu) with a signed divide (div).

In order to fix the rounding problem, more work was needed.

The coded formula is:

```
mean = (sum + (count/2))/count
```

In practice, the MIPS assembler implements divides as an embedded assembler macro which performs a divide by zero check. In the case of Pixel3x3 it is as follows:

```
0370 2000638E    lw      $3,32($19)
0374 00000000
0378 42100300    srl     $2,$3,1
037c 2400648E    lw      $4,36($19)
0380 00000000

---- Code we're going to muck with ----
0384 21104400    addu   $2,$2,$4
0388 1B004300    divu   $2,$2,$3
      02006014
      00000000
      0D000700
---- End of code we're going to muck with ----
0398 12100000
039c 00000000
      00000000
03a4 280062AE    sw     $2,40($19)

...

```

Since the C++ code already has an earlier zero check on the denominator, the patch re-codes this portion function as follows:

```
0370 2000638E    lw      $3,32($19)
0374 00000000
0378 42100300    srl     $2,$3,1
037c 2400648E    lw      $4,36($19)
0380 00000000

---- Start of change ----
0384          bgez   $4,positive
0388          add    $2,$2,$4
038c          sub    $2,$2,$3
positive:
0390          div    $0,$2,$3
0394          nop
---- End of change ----

0398 12100000
039c 00000000
      00000000
03a4 280062AE    sw     $2,40($19)

```

Applicable Reports/Requests:
SPR-128

Test Results:
reproduce --> PASS
fix --> PASS

Replaced Functions:

Command Impact:
None.

Telemetry Impact:

None.

Science Impact:

Without this patch, the corner mean values in Graded Telemetry mode may occasionally be invalid. There is a deterministic ground algorithm which can detect and correct for this effect, but without the flight patch or the ground algorithm, the corner mean values may be grossly incorrect in some cases.

Once the patch is in place, the corner mean values should be within 1/2 an ADU of the true mean, regardless of sign, without further action needed by the ground science software.

=====
Patch Name: buscrash

Part Number: 36-58030.30

Version: A

SCO:

Description:

Reason:

If ACIS is computing bias maps when commanded to power down its front-end processors (FEPs), it is likely to crash the back-end processor (BEP) interface bus, causing the BEP to reboot without flight software patches. Normal operations must be restored via ground command. The cause of the problem has been traced to a design flaw in the BEP flight software and this ECO describes a small patch that will fix it.

Symptom:

During execution of SCS107, typically due to high background radiation, ACIS is powered down. Science telemetry reports that the flight s/w version number is 11, whereas typical values (depending in the patch combination) are 30 or higher, indicating that the BEP rebooted itself. Subsequent inspection of the recorded telemetry shows no scienceReport packet from the last science run, but a bepStartupMessage packet with lastFatalCode=7 and watchdogFlag=1.

Symptom Impact:

Since the observatory is usually in safe mode for several hours following the SCS107, there is generally sufficient time to establish a realtime contact, set the BEP's warm-boot flag, and restart it. However, this takes time and manpower.

Symptom Cause:

The bus crash has been traced to a flaw in the FepManager::loadBadPixel() method. This routine is executed after the FEP bias maps have been created and before they are (optionally) reported in telemetry. It uses the memory-mapped interface between BEP and FEP to change those locations in the FEP bias maps that correspond to "bad" pixels or whole columns. However, unlike all other FepManager operations, loadBadPixel() does not confirm that a FEP is powered up before it writes to its map. This causes the bus crash.

Fix Description:

Call the FepManager::isEnabled() method to check if the FEP is powered up before writing to a FEP's bias memory (and parity plane).

Applicable Reports/Requests:

SPR-140

Test Results:

reproduce --> PASS

fix --> PASS

Replaced Functions:

FepManager::loadBadPixel

Command Impact:

None.

Telemetry Impact:
None.

Science Impact:
None.

=====
Patch Name: buscrash2

Part Number: 36-58030.30

Version: B-

SCO:

Description:

Reason:

If ACIS is copying bias maps to telemetry when commanded to power down its front-end processors (FEPs), it is likely to crash the back-end processor (BEP) interface bus, causing the BEP to reboot without flight software patches. Normal operations must be restored via ground command. The cause of the problem has been traced to a design flaw in the BEP flight software and this ECO describes a small patch that will fix it.

Symptom:

During execution of SCS107, typically due to high background radiation, ACIS is powered down. Science telemetry reports that the flight s/w version number is 11, whereas typical values (depending in the patch combination) are 30 or higher, indicating that the BEP rebooted itself. Subsequent inspection of the recorded telemetry shows no scienceReport packet from the last science run, but a bepStartupMessage packet with lastFatalCode=7 and watchdogFlag=1.

Symptom Impact:

Since the observatory is usually in safe mode for several hours following the SCS107, there is generally sufficient time to establish a realtime contact, set the BEP's warm-boot flag, and restart it. However, this takes time and manpower.

Symptom Cause:

The bus crash has been traced to a flaw in the BiasThief::checkMonitor() method. This routine is executed after the FEP bias maps have been created and it copies them to telemetry. It uses the memory-mapped interface between BEP and FEP to access the maps but, unlike other FepManager operations, it does not confirm that a FEP is powered up before it reads the maps. This causes the bus crash.

Fix Description:

Call the FepManager::isEnabled() method to check if the FEP is powered up before reading from a FEP's bias memory. This is done by patching BiasThief::checkMonitor() as follows:

```
class Test2_BiasThief : public BiasThief
{
public:
    Boolean checkMonitor(FepId fepid);
};

Boolean Test2_BiasThief::checkMonitor(FepId fepid)
{
    DebugProbe probe;
    Boolean retval = BoolTrue;    // Assume no abort

    if (fepid >= FEP_COUNT ||
        fepManager.isEnabled (fepid) == BoolFalse) {
        swHousekeeper.report(SWSTAT_FEPREC_POWEROFF, fepid);
        retval = BoolFalse;    // FEP not available or powered
    } else {
        unsigned caught = requestEvent (EV_TASKQUERY | EV_ABORT);
```

```
        if (caught & EV_TASKQUERY) {
            taskMonitor.respond ();
        }
        if (caught & EV_ABORT) {
            retval = BoolFalse;
        }
    }
    // ---- Return BoolTrue if no abort, BoolFalse if aborted ----
    return retval;
}
```

To pass the fepId as an argument to this version of checkMonitor(), other BiasThief methods are patched inline, as follows:

```
biasthief+0x0340:
    sw      $6,36($sp)

biasthief+0x0360:
    lw      $5,36($sp)

biasthief+0x04d4:
    lw      $5,104($sp)

biasthief+0x050c:
    lw      $6,104($sp)

biasthief+0x07b0:
    move    $5,$18

biasthief+0x07f4:
    move    $6,$18
```

Applicable Reports/Requests:
SPR-142

Test Results:
reproduce --> PASS
fixTe --> PASS
fixCc --> PASS

Replaced Functions:
BiasThief::checkMonitor

Command Impact:
None.

Telemetry Impact:
None.

Science Impact:
None.

=====
Patch Name: rquad

Part Number: 36-58030.14
Version: A
SCO: 36-1000

Description:

Reason:

This patch fixes software problem report SPR-121.

Symptom:

If the center pixel of a 3x3 event is in the last column of any but the right-most quadrant (i.e. in FULL mode, quadrants A, B or C, but not D), the flight software will inappropriately use the delta overclock and split threshold for the center pixel's quadrant on the pixels on the right edge of the event. The instrument is supposed to use the delta overclock and split thresholds for the next quadrant on these pixels.

Symptom Impact:

This may lead to an incorrect estimate of the event's total pulse height and grade, possibly leading to inappropriate pulse height and grade filtering of these events, or, when using Graded Event formats, incorrect pulse height and grade code values.

Symptom Cause:

The flight software is fetching the quadrant identifier for the wrong column position for the right edge pixels:

```
quad = exposure->getQuadrant (col);  
doclk[1] = exposure->getOverclockDelta (quad);  
split[1] = exposure->getSplitThreshold (quad);
```

```
WRONG---> quad = exposure->getQuadrant (col);  
doclk[2] = exposure->getOverclockDelta (quad);  
split[2] = exposure->getSplitThreshold (quad);
```

```
computePhGrade (doclk, split);
```

This should be:

```
quad = exposure->getQuadrant (col);  
doclk[1] = exposure->getOverclockDelta (quad);  
split[1] = exposure->getSplitThreshold (quad);
```

```
CORRECT---> quad = exposure->getQuadrant (col+1);  
doclk[2] = exposure->getOverclockDelta (quad);  
split[2] = exposure->getSplitThreshold (quad);
```

```
computePhGrade (doclk, split);
```

Fix Description:

The patch increments the column register variable using an "nop" slot of an earlier instruction following the previous call to exposure->getQuadrant() and prior to the last call to exposure->getQuadrant().

This is the last time the register is used in the function, so it won't corrupt subsequent code, and the "nop" was inserted by the compiler after a "lw", which allows for increments of registers unrelated to the "lw".

```

                                05cc 2C00A2AF          sw      $2,44($sp)
                                $LM84:
                                210:../filesscience/pixel3x3.C ****
                                211:../filesscience/pixel3x3.C ****      quad = exposure->getQ
uadrant (col);
                                05d0 5400028E          lw      $2,84($16)
"addu $18,$18,1" --->> 05d4 00000000
                                05d8 0800428C          lw      $2,8($2)
                                00000000
                                05e0 21200002          move    $4,$16
                                .set    noreorder
                                .set    nomacro
"col" is passed in      05e4 09F84000          jal    $31,$2
a delay slot          --->>05e8 21284002          move    $5,$17
                                .set    macro
                                .set    reorder

                                05ec 21884000          move    $17,$2
                                $LM85:
                                ../filesscience/pixel3x3.C ****      doclk[2] = exposure->getO
verclockDelta (quad);
                                05f0 5400028E          lw      $2,84($16)
                                05f4 00000000
                                05f8 0400428C          lw      $2,4($2)
                                00000000
                                0600 21200002          move    $4,$16
                                .set    noreorder
                                .set    nomacro
                                0604 09F84000          jal    $31,$2
                                0608 21282002          move    $5,$17
                                .set    macro
                                .set    reorder

                                060c 2000A2AF          sw      $2,32($sp)
                                $LM86:
                                ../filesscience/pixel3x3.C ****      split[2] = exposure->getS
plitThreshold (quad);
                                .stabn 68,0,213,$LM86
                                0610 5400028E          lw      $2,84($16)
                                0614 00000000
                                0618 0C00428C          lw      $2,12($2)
                                00000000
                                0620 21200002          move    $4,$16
                                .set    noreorder
                                .set    nomacro
                                0624 09F84000          jal    $31,$2
                                0628 21282002          move    $5,$17
                                .set    macro
                                .set    reorder

                                062c 3000A2AF          sw      $2,48($sp)
                                $LM87:
                                ../filesscience/pixel3x3.C ****
                                ../filesscience/pixel3x3.C ****      computePhGrade (doclk, sp
lit);
                                .stabn 68,0,215,$LM87
                                0630 1000828E          lw      $2,16($20)
                                0634 00000000
                                0638 1C00428C          lw      $2,28($2)

```



```
00000000
0640 21208002          move    $4,$20
0644 1800A527          addu   $5,$sp,24
                          .set   noreorder
                          .set   nomacro
0648 09F84000          jal   $31,$2
064c 2800A627          addu   $6,$sp,40
                          .set   macro
                          .set   reorder
```

```
$LBB29:
$LM88:
$LBB30:
$LBE30:
$LM89:
$LBE29:
$LM90:
```

```
../filesscience/pixel3x3.C ****
../filesscience/pixel3x3.C **** //
../filesscience/pixel3x3.C **** }
```

```
$LBE26:
```

```
0650 4C00BF8F          lw     $31,76($sp)
      00000000
0658 4800B48F          lw     $20,72($sp)
      00000000
0660 4400B38F          lw     $19,68($sp)
      00000000
0668 4000B28F          lw     $18,64($sp)
      00000000
0670 3C00B18F          lw     $17,60($sp)
      00000000
0678 3800B08F          lw     $16,56($sp)
      00000000
0680 5000BD27          addu   $sp,$sp,80
0684 0800E003          j      $31
      00000000
```

```
.end Pixel3x3::attachData(FEEven
```

```
tRec3x3 const *, EventExposure *)
```

```
$LM91:
```

Applicable Reports/Requests:
SPR-121

Test Results:
reproduce --> PASS
fix --> PASS

Replaced Functions:

Command Impact:
None

Telemetry Impact:
See SCIENCE IMPACT.

Science Impact:
Without this patch, all Timed Exposure and CC3x3 events on the left edge of a quadrant boundary may have incorrect pulse heights and

grades, and events which impact at these positions may be inappropriately filter out or telemetered if pulse height and grade filters are used.

=====
Patch Name: condock

Part Number: 36-58030.17

Version: A

SCO: 36-1012

Description:

Reason:

The first timed exposure frames received during OAC (e.g., SOP_61052_DARK_CUR) showed sporadic increases in the overclock averages, and anomalous dark patches within bias maps. Once raw frames were examined (in SOP_61054_RAW_DATA and SAP_61079_RAW_BIAS), the effect was seen to be caused by charged particle background "leaking" into the overclocks.

Fix Description:

Patch the FEP overclock processing function, fepOclkProc in fep/fepCtl.c, to "condition" the overclock sum on a row-by-row basis. The patch, which will not apply to OC_RAW or OC_HIST modes, will ignore the overclock sum of particular row and node if it exceeds the previous sum by some suitable threshold. This entails replacing the following fepOclkProc() code:

```
for (ioclck = 0; ioclck < fp->tp.noclck; ioclck++) {
    unsigned p0 = *fp->oc.optr++;
    unsigned p1 = *fp->oc.optr++;
    switch (fp->tp.quadcode) {
    case FEP_QUAD_AC:
        fp->oc.osum[0] += PIXEL0(p0) & PIXEL_MASK;
        fp->oc.osum[1] += PIXEL0(p1) & PIXEL_MASK;
        break;
    case FEP_QUAD_BD:
        fp->oc.osum[0] += PIXEL1(p0) & PIXEL_MASK;
        fp->oc.osum[1] += PIXEL1(p1) & PIXEL_MASK;
        break;
    default:
        fp->oc.osum[0] += PIXEL0(p0) & PIXEL_MASK;
        fp->oc.osum[1] += PIXEL1(p0) & PIXEL_MASK;
        fp->oc.osum[2] += PIXEL0(p1) & PIXEL_MASK;
        fp->oc.osum[3] += PIXEL1(p1) & PIXEL_MASK;
        break;
    } /* end switch */
} /* end for ioclck */
```

with an inline patch that saves R9-R12:

```
condockCtl(fp);

    subu    $sp,$sp,16
    sw     $9,0($sp)
    sw     $10,4($sp)
    sw     $11,8($sp)
    sw     $12,12($sp)
    jal    condockCtl
    move   $4,$16
    lw     $9,0($sp)
    lw     $10,4($sp)
    lw     $11,8($sp)
    lw     $12,12($sp)
    j      fepCtl+0x0f74
    addu   $sp,$sp,16
```

and adding the condockCtl function:

```
void condockCtl(FEPparm *fp)
{
    unsigned dsum = OCLK_COND * fp->tp.noclk;
    unsigned ioclk, iquad;

    /* clear local accumulator */
    for (iquad = 0; iquad < 4; iquad++) {
        fp->oc.ossql[iquad] = 0;
        /* clear saved row sum at start of frame */
        if (fp->oc.osum[iquad] == 0) {
            fp->oc.ossqh[iquad] = 0;
        }
    } /* end for iquad */

    /* accumulate the overclock sums */
    for (ioclk = 0; ioclk < fp->tp.noclk; ioclk++) {
        unsigned p0 = *fp->oc.optr++;
        unsigned p1 = *fp->oc.optr++;
        switch (fp->tp.quadcode) {
            case FEP_QUAD_AC:
                fp->oc.ossql[0] += PIXEL0(p0) & PIXEL_MASK;
                fp->oc.ossql[1] += PIXEL0(p1) & PIXEL_MASK;
                break;
            case FEP_QUAD_BD:
                fp->oc.ossql[0] += PIXEL1(p0) & PIXEL_MASK;
                fp->oc.ossql[1] += PIXEL1(p1) & PIXEL_MASK;
                break;
            default:
                fp->oc.ossql[0] += PIXEL0(p0) & PIXEL_MASK;
                fp->oc.ossql[1] += PIXEL1(p0) & PIXEL_MASK;
                fp->oc.ossql[2] += PIXEL0(p1) & PIXEL_MASK;
                fp->oc.ossql[3] += PIXEL1(p1) & PIXEL_MASK;
                break;
        } /* end switch */
    } /* end for ioclk */

    /* condition the sums */
    for (iquad = 0; iquad < 4; iquad++) {
        if (fp->oc.ossqh[iquad] == 0) {
            /* always save first row sum */
            fp->oc.ossqh[iquad] = fp->oc.ossql[iquad];
        } else if (fp->oc.osum[iquad] == fp->oc.ossqh[iquad] &&
            fp->oc.ossqh[iquad] > fp->oc.ossql[iquad] + dsum) {
            /* if second row sum much less than first, replace the
            total sum by twice the second sum */
            fp->oc.osum[iquad] = fp->oc.ossqh[iquad] = fp->oc.ossql[iquad];
        } else if (fp->oc.ossql[iquad] <= fp->oc.ossqh[iquad] + dsum) {
            /* save row sum if not much greater than the saved sum */
            fp->oc.ossqh[iquad] = fp->oc.ossql[iquad];
        }
        /* increment overclock accumulator */
        fp->oc.osum[iquad] += fp->oc.ossqh[iquad];
    } /* end for iquad */
}
```

The algorithm uses the oc.ossql[4] and oc.ossqh[4] fields which would not otherwise participate in OC_SUM mode, and whose prior contents may be safely overwritten. The oc.ossql fields are used to accumulate the overlocks of the current row, and the current "best" value of this

sum is saved from row to row in oc.ossqh. If the current row sum exceeds the current best sum by a constant OCLK_COND times the number of overclocks in the row, the current best sum will be used in its place; otherwise, the sum of the current row will replace the current best. The first two rows of each frame receive special treatment: the first row sum is used to initialize oc.ossqh -- the "best" sum -- and, if the sum of the second row is anomalously LOWER than this, the best row sum and the running total sum are corrected.

Applicable Reports/Requests:
SPR-127

Test Results:
reproduce --> PASS
fix --> PASS

Replaced Functions:

Command Impact:
None

Telemetry Impact:
None

Science Impact:
With this patch installed, the effect of background events on overclock averages will be greatly reduced, directly reducing systematic errors within bias maps and increasing the accuracy of photon energy determination.

=====
Patch Name: histogrammean

Part Number: 36-58030.15

Version: A

SCO: 36-996

Description:

Reason:

In raw TE histogram mode, the FEPs report the mean of each CCD quadrant's overclocks. This is done in two steps: first, the overclocks of each quadrant of each frame are summed into fields "oc.osum" in the FEpparm structure, and these are then averaged over the separate "histogramCount" frames and reported to the BEP in "omean" fields in FEPEventRecHist structures. The error is caused by using the 16-bit "omean" fields as accumulators, as well as final values, since, if the mean overclock value multiplied by "histogramCount" exceeds 65535, overflow will occur.

Fix Description:

The patch adds 8 32-bit integer fields to the end of the D-cache stack employed by the fepCtl function. Within FEPsciTimedHist, machine instructions are altered to initialize these fields to zero, to use them to accumulate the intermediate sums, and hence to form the means which are stored into "omean".

(a) increase fepCtl stack length by an extra 32 bytes

```

                .globl fepCtl_lst_0000_0000
                .ent   fepCtl_lst_0000_0000
fepCtl_lst_0000_0000:
0000 88FABD27          subu    $sp,$sp,1368+32
0004 5405BFAF
                .end   fepCtl_lst_0000_0000

```

(b) decrease fepCtl stack length by an extra 32 bytes

```

                .globl fepCtl_lst_012c_012c
                .ent   fepCtl_lst_012c_012c
fepCtl_lst_012c_012c:
0128 00000000
012c 7805BD27          addu    $sp,$sp,1368+32
0130 0800E003
                .end   fepCtl_lst_012c_012c

```

(c) set mean and variance sums to zero

```

                .globl fepSciTimed_lst_1858_1864
                .ent   fepSciTimed_lst_1858_1864
fepSciTimed_lst_1858_1864:
1854 80180B00          addu    $3,$3,$16
1858 21187000          sw     $0,1368-16($3)
185c 480560AC          sw     $0,1368($3)
1860 580560AC          sh     $0,20($2)
1864 140040A4
1868 0C0044A4
                .end   fepSciTimed_lst_1858_1864

```

(d) increment mean sum

```
                .globl  fepSciTimed_lst_lacc_ladc
                .ent    fepSciTimed_lst_lacc_ladc
fepSciTimed_lst_lacc_ladc:
lab0 1B006A00
      02004015
      00000000
      0D000700
      12180000
lacc 34050925      addu   $9,$8,1368-36
lad0 4805028D      lw     $2,1368-16($8)
lad4 00000000      nop
lad8 21104300      addu   $2,$2,$3
ladc 480502AD      sw     $2,1368-16($8)
lae0 1B00AA01
lae4 02004015
lae8 00000000
laec 0D000700
laf0 12200000
                .end    fepSciTimed_lst_lacc_ladc
```

(e) save stack pointer in R9

```
                .globl  fepSciTimed_lst_lc38_lc38
                .ent    fepSciTimed_lst_lc38_lc38
fepSciTimed_lst_lc38_lc38:
lc34 1403028E
lc38 48050926      addu   $9,$16,1368-16
lcec 22004010
                .end    fepSciTimed_lst_lc38_lc38
```

(f) load overclock mean sum

```
                .globl  fepSciTimed_lst_lc50_lc50
                .ent    fepSciTimed_lst_lc50_lc50
fepSciTimed_lst_lc50_lc50:
lc4c 21187200
lc50 0000228D      lw     $2,0($9)
lc54 00000000
                .end    fepSciTimed_lst_lc50_lc50
```

(g) load overclock variance sum

```
                .globl  fepSciTimed_lst_lc84_lc84
                .ent    fepSciTimed_lst_lc84_lc84
fepSciTimed_lst_lc84_lc84:
lc80 21187200
lc84 1000228D      lw     $2,16($9)
lc88 00000000
                .end    fepSciTimed_lst_lc84_lc84
```

(h) increment R9

```
                .globl  fepSciTimed_lst_lcb8_lcb8
                .ent    fepSciTimed_lst_lcb8_lcb8
fepSciTimed_lst_lcb8_lcb8:
lcb4 1403028E
lcb8 04002925      addu   $9,$9,4
lcbc 2B106201
                .end    fepSciTimed_lst_lcb8_lcb8
```

SPR-123

Test Results:

```
reproduce --> PASS
fix --> PASS
```

Replaced Functions:

Command Impact:

None

Telemetry Impact:

None. It should be pointed out that an alternative approach to fixing this problem is to add the following code to the downlink raw histogram software, although this algorithm may fail for very large values of "histogramCount".

```
if (fs->meanOverclock[node] < fs->minimumOverclock[node] ||
    fs->meanOverclock[node] > fs->maximumOverclock[node]) {
    unsigned hh = loadTeBlock_histogramCount(param);
    double dmlim = 8192.0*hh*loadTeBlock_overclockPairsPerNode(param);
    unsigned mmlim, mlim = (dmlim < 0x7fffffff) ? dmlim : 0x7fffffff;
    for (mm = 0; mm < mlim; mm += 65536) {
        unsigned nn = fs->meanOverclock[node]+(mm+hh/2)/hh;
        if (nn >= fs->minimumOverclock[node] &&
            nn <= fs->maximumOverclock[node]) {
            fs->meanOverclock[node] = nn;
            break;
        }
    }
}
```

Science Impact:

None -- raw histogram mode is not necessary for science processing.

TITLE: ACIS Flight Software Optional Patch Component Release Notes

DOCUMENT NUMBER: 36-58020 REVISION: F

ORIGINATOR: Peter G. Ford <pgf@space.mit.edu>

LETTER	SCO NO.	DESCRIPTION	APPROVED	DATE
01	36-987	Initial numeric release	jimf	11/12/1998
A	36-1007	Bug fixes, incorporate tests	RFG	05/12/1999
B	36-1019	Add new patches, retest	RFG	12/16/1999
C	36-1022	Add new patches, retest	RFG	03/21/2003
D	36-1040	Add new patches, retest	RFG	09/29/2009
E	36-1042	No new patches, retest	RFG	01/06/2010
F	36-1044	Add txings patch, retest	RFG	03/02/2011
F	36-1044	Add txings patch		

=====
Title: ACIS Optional Patch Release Notes for Version F

Software Change Order: 36-1044

Build Date: Sun May 22 18:42:31 EDT 2011
Part Number: 36-58020
Version: F
CVS Tag: release-E-opt-F

Std Number: 36-58010
Std Version: E
Std Tag: release-E
Std SCO: 36-1042

IPCL Number: 36-53204.0204
IPCL Version: N
IPCL CVS Tag: release-N

Description:

This is the sixth letter release of the optional patch set for the ACIS Flight Software. The purpose of this release is to add the txings and fepthrottle patches and test them with the Rev. E Standard Patch release.

Although the patches listed in this release have been tested in combination with the standard patch release, they have NOT been tested in various combinations with each other as part of this release. Each needed combination will be provided a distinct part number, and will be released individually, based on the patches provided in this release.

This release consists of the following optional flight patches:

cc3x3	- Continuous Clocking 3x3 Event Mode
ccignore	- Ignore Continuous Clocking data frames
compressall	- Fixes SPR 134
ctireport1	- Reports precursor charge
ctireport2	- Reports precursor charge
eventhist	- Timed Exposure Event Histogram Mode
reportgradel	- Addresses SPR 132
smtimedlookup	- Supports eventhist and ctireport
teignore	- Ignore Timed Exposure data frames
untricklebias	- Fixes SPR 133
* txings	- Triggers bilevels on excess threshold crossings

This release also contains a set of informally controlled engineering patches, used for ground testing, debugging and experimentation:

hybrid	- Prototype of a hybrid clocking mode
squeegy	- Prototype of a squeegee clocking mode
fepbiasparity1	- Prototype of the fepbiasparity2 patch
forcebiastrickle	- Patch to set trickleBias flag
tlmio	- Telemetry Standard I/O Utility Routines
printswhouse	- Print S/W Housekeeping reports in realtime
deaeng	- Detect/configure for DEA Engineering video boards
dearepl	- Stubs for use when a DEA is not attached
* fepthrottle	- Reduces FEP event candidates

Addressed Problem Reports:
SPR-134

SPR-126
SPR-132
SPR-133
SPR-120
SPR-124

Included Patches:

cc3x3 (4636 bytes)
ccignore (36 bytes)
compressall (2368 bytes)
ctireport1 (5452 bytes, depends on smtimedlookup)
ctireport2 (2784 bytes, depends on smtimedlookup)
deaeng (2604 bytes, depends on tlmio, conflicts with dearepl)
dearepl (556 bytes, conflicts with deaeng)
eventhist (5908 bytes, depends on smtimedlookup)
printshouse (7240 bytes, depends on tlmio)
reportgradel (816 bytes)
smtimedlookup (3712 bytes)
teignore (36 bytes)
tlmio (10312 bytes)
txings (3128 bytes)
untricklebias (1740 bytes, depends on buscrash2)

=====
Patch Name: reportgradel

Part Number: 36-58030.22
Version: A
SCO: 36-1021
Environment: flight

Conflicts:
Depends On:
Size: 816 bytes

Bcmd File: opt_reportgradel.bcnd
Pkts File: opt_reportgradel.pkts

Description:

This patch reports per-FEP event filtering statistics via software housekeeping. The SwHousekeeper constructor is patched in order to add an extra 54 housekeeping codes, 9 per FEP, as follows:

```
SW_FILT_NONE,      /* events unfiltered */
SW_FILT_ENERGY,   /* events filtered by energy */
SW_FILT_GRADE1,   /* events filtered by SW_GRADE_CODE1 */
SW_FILT_GRADE2,   /* events filtered by SW_GRADE_CODE2 */
SW_FILT_GRADE3,   /* events filtered by SW_GRADE_CODE3 */
SW_FILT_GRADE4,   /* events filtered by SW_GRADE_CODE4 */
SW_FILT_GRADE5,   /* events filtered by SW_GRADE_CODE5 */
SW_FILT_OTHER,    /* events filtered by other grade */
SW_FILT_WIN,      /* events filtered by window */
```

These SwStatistic codes begin at a value of SWSTAT_FILTER_BASE. They are defined in "acis_h/interface.h", along with the 5 special grade codes:

```
SW_GRADE_CODE1 = 24,
SW_GRADE_CODE2 = 66,
SW_GRADE_CODE3 = 107,
SW_GRADE_CODE4 = 214,
SW_GRADE_CODE5 = 255
```

Thus, the number of grade 214 events rejected by FEP_3 during the current housekeeping interval will be reported in swHousekeeping packets with a "statistics[].swStatisticId" value of SWSTAT_FILTER_BASE+SW_FILT_GRADE4+(9*FEP_3). The corresponding "statistics[].count" field will contain the number of events in this particular class from this particular FEP during the current ~64 sec housekeeping interval. As an aide to synchronizing housekeeping data and event packets, the "statistics[].value" field will contain the most recent exposure number read from this FEP during this interval.

Applicable Reports/Requests:
SPR-132

Test Results:
testTe --> PASS
testCc --> PASS

Replaced Functions:

PmEvent::filterEvent

Command Impact:

None.

Telemetry Impact:

No reduction of telemetry throughput is anticipated. To identify the new housekeeping fields, ground software must recognize the new SwStatistic codes. Refer to the ACIS Software IP&CL Release Notes, Rev. L or later, for details

Science Impact:

None.

=====
Patch Name: untricklebias

Part Number: 36-58030.28
Version: B
SCO: 36-1028
Environment: flight

Conflicts:
Depends On: buscrash2
Size: 1740 bytes

Bcmd File: opt_untricklebias.bcnd
Pkts File: opt_untricklebias.pkts

Description:

For reasons unknown, the BEP has occasionally run the science and bias thief tasks simultaneously. This causes the FEPs to start searching for x-ray events while the BEP is copying their bias maps to telemetry. If the threshold crossing frequency is sufficiently high, this can trigger an error in the FEP firmware leading to a "T-plane latchup" condition.

The untricklebias patch prevents this behavior by ensuring that the FEP bias maps are never accessed by the BiasThief task. Instead, the science task is given these functions.

The main routine of the bias thief task is repaced by Test_BiasThief::goTaskEntry, which does nothing beyond waking up whenever the task monitor tells it to, but goes back to sleep again immediately.

Where necessary, the remaining BiasThief methods that are called from the science task are replaced by methods that do not notify the bias thief task that a change has been made. The trickleTeBias and trickleCcBias do not need to be patched, but the checkMonitor method must be replaced with a version that is appropriate for being called from the science task. Note that it tests the EV_SM_BIAS_ABORT_RUN in the event mask: this is the value appropriate for a science task abort.

When used with Standard Patch Release D or higher, containing the buscrash2 patch, the BiasThief::checkMonitor() method has been updated to test whether the fepId is powered up. This method must therefore override both the original checkMonitor() and the updated version loaded by the buscrash2 patch.

Applicable Reports/Requests:
SPR-133

Test Results:
patchTe --> PASS
patchAll --> PASS
patchCc --> PASS

Replaced Functions:
BiasThief::abort

```
ScienceMode::waitForBiasTrickle  
BiasThief::goTaskEntry  
BiasThief::biasReady  
BiasThief::checkMonitor
```

Command Impact:
None.

Telemetry Impact:
None.

Science Impact:
None.

=====
Patch Name: deaeng

Part Number: 36-58030.11
Version: 02
SCO: 36-1010
Environment: engineering

Conflicts: dearepl
Depends On: tlmio
Size: 2604 bytes

Bcmd File: opt_deaeng.bcnd
Pkts File: opt_deaeng.pkts

Description:

This patch provides the basic capability to detect and communicate with the engineering version of the DEA CCD controller boards. For historical reasons, these boards have a different interface than the flight CCD controllers.

This patch relies on printf() being installed (see tlmio).

Applicable Reports/Requests:
TOOL-PENDING

Test Results:
No Tests Specified

Replaced Functions:
DeaCcdController::updateRegister
DeaCcdController::powerOn
DeaCcdController::writeData

Command Impact:
This patch will determine the type of video boards installed in the system. Due to the interface differences between boards, high-speed tap commands will not work on engineering video boards, but will continue to work on "flight-like" video boards.

Telemetry Impact:
Since this patch calls printf(), it will result in TTAG_USER telemetry packets.

Science Impact:
N/A

=====
Patch Name: cc3x3

Part Number: 36-58030.06
Version: B
SCO: 36-1018
Environment: flight

Conflicts:
Depends On:
Size: 4636 bytes

Bcmd File: opt_cc3x3.bcmd
Pkts File: opt_cc3x3.pkts

Description:

This patch implements the Continuous Clocking 3x3 Event Mode. In this mode, the instrument performs the standard continuous clocking manipulation of the CCDs, but rather than accept and telemetry 1x3 events, the mode processes 3x3 event islands, improving the spectral performance of the mode and reducing the problems associated with vertically split events.

Because the Continuous Clocking parameter block only provides 4 bits for defining the grade selection for the mode (in 1x3, only 4 bits were necessary), this patch provides table which maps the 4-bit code into a set of pre-built 256-bit grade selection masks. In this release, the grade selection map is populated with masks provided by Fred Baganoff. Refer to grade_table.html for a description of the grade families. The following table summarizes the selections:

- Code 0 - Reject all grades
- Code 1 - Reject ASCA grades 1,2,3,4,5,6,7
- Code 2 - Reject ASCA grades 1,5,6,7
- Code 3 - Reject ASCA grades 1,5,7
- Code 4 - Undefined (currently rejects all grades)
- Code 5 - Undefined (currently rejects all grades)
- Code 6 - Undefined (currently rejects all grades)
- Code 7 - Reject ACIS flight grades 24,66,107,127,214,223,248,251,254,255
- Code 8 - Reject ACIS flight grades 24,107,127,214,223,248,251,254,255
- Code 9 - Reject ACIS flight grades 24,66,107,214,248,255
- Code 10 - Reject ACIS flight grades 24,66,107,214,255
- Code 11 - Reject ACIS flight grades 24,107,214,248,255
- Code 12 - Reject ACIS flight grades 24,107,214,255
- Code 13 - Reject ASCA grade 7
- Code 14 - Reject ACIS flight grade 255
- Code 15 - Accept all grades

NOTE: CC3x3 Codes 0 and 15 have the same effect as their numerical equivalents in CC1x3, where 0 will reject all events, and 15 will accept events with any grade code.

Applicable Reports/Requests:
SPR-126
SPR-120
SPR-124

Test Results:

```
unit --> PASS
smoke --> PASS
```

Replaced Functions:

```
SmContClocking::setupFepBlock
SmContClocking::setupProcess
SmContClocking::terminate
```

Command Impact:

This version of CC3x3 uses different grade sets than the previous version. This may have an impact on the grade selection field of CC Parameter Block command packets already built for CC3x3 observations.

This mode is invoked by using the FEP_CC_MODE_EV3x3 (2) in the fepMode field of the Continuous Clocking Parameter block, in conjunction with any of the BEP_CC event processing modes for the bepPackingMode field. This restricts the use of this mode to CC Faint and CC Graded modes. This patch does NOT support other Timed Exposure derived modes, such as Faint with Bias, 5x5, nor any of the existing nor patched histogram modes.

At the onset of a CC3x3 science run, the run will force two resets and reloads of the FEP software, the first to ensure that the boot-strap code is in the FEPs, and the second to load the patch code into the FEPs. This will always add up to 14 seconds per FEP to the start-up time of the run, compared to runs where the FEPs were already loaded and running.

To ensure that the patch is not present at the start of the next run, which may or may not be a CC3x3 run, a CC3x3 science run will always force the FEPs into a reset state at the end of the run. This will add another 7 seconds per FEP to the start up time of the run following a CC3x3 run, relative to the normal start up time, where the FEPs were already loaded and running.

These resets will also impact the power consumption of ACIS, where the system will draw up to 16 watts less than normal (with all 6 on and running) while the FEPs are held a reset state.

Refer to the ACIS Software IP&CL Structure Definitions, Rev. L or later for details.

Telemetry Impact:

This mode defines 4 new telemetry packet types.

When configured for FEP_CC_MODE_EV3x3 and BEP_CC_MODE_FAINT, the patch produces TTAG_SCI_CC_REC_FAINT3x3 exposure records and TAG_SCI_CC_DAT_FAINT3x3 event data packets.

When configured for FEP_CC_MODE_EV3x3 and BEP_CC_MODE_GRADED, it produces TTAG_SCI_CC_REC_GRADED3x3 exposure records and TTAG_SCI_CC_DAT_GRADED3x3 event data packets.

The size of and overhead of these packets are the same as their Timed Exposure counterparts, TTAG_SCI_TE_REC_FAINT3x3, TTAG_SCI_TE_DAT_FAINT3x3, TTAG_SCI_TE_REC_GRADED3x3 and TTAG_SCI_TE_DAT_GRADED3x3.

When used, a CC3x3 science run will produce additional Software Housekeeping counts to the FEP write and execute statistics, reflecting the additional resets and reloads of the FEPs. Runs immediately following a CC3x3 run will also produce additional FEP related counts, as they load and run the reset FEPs.

Refer to the ACIS Software IP&CL Structure Definitions, Rev. L or later for details

Science Impact:

This version of CC3x3 uses different grade sets than the previous version. The ground data analysis software may have to be aware of which version of CC3x3 is installed for a given set of CC3x3 data. Please refer to the ACIS command generation system for the set of ACIS Software Version identifiers (telemetered in the BEP Startup Message and in each Software Housekeeping telemetry packet) corresponding to the different installed CC3x3 versions.

This mode produces a new type of data product, consisting of 3x3 islands around accepted events in Continuous Clocking mode. This is intended to provide better spectral resolution and event detection performance when in Continuous Clocking mode.

This mode will not report events on row 0 and row 511, leaving a 2-row timing gap with a period of 512 rows.

As in other Continuous Clocking modes, no bias errors will be reported when in this mode, since the bias map is extremely redundant (there's 512 copies of the bias value for any given column).

=====
Patch Name: tlmio

Part Number: 36-58030.07
Version: 02
SCO: 36-1010
Environment: flight

Conflicts:
Depends On:
Size: 10312 bytes

Bcmd File: opt_tlmio.bcmd
Pkts File: opt_tlmio.pkts

Description:

This patch provides basic standard I/O functions which emit TTAG_USER telemetry packets containing data written via calls to write().

This patch stubs the functions open(), close() and read(), and implements the function write(), used by higher level I/O library functions, such as printf().

The patch maintains a 1024 word telemetry buffer just at the end of bulk memory. write() appends data to this buffer until either the buffer fills, or until a newline is written. Once write() fills the buffer or a newline is encountered, the telemetry buffer is sent as follows:

1. Interrupts are disabled
2. The hardware is polled until the current packet is finished.
3. The packet buffer header is filled in, and the first data word is set to 0 (a hook used to support different subtypes of TTAG_USER).
4. Transfer the packet
5. Wait for the transfer to complete
6. If no transfer was in progress prior to the interrupt disable, clear the pending interrupt caused by the TTAG_USER packet transfer
7. Reset the the buffer contents
8. Reenable interrupts

Applicable Reports/Requests:
TOOL-PENDING

Test Results:
No Tests Specified

Replaced Functions:

Command Impact:
None

Telemetry Impact:
If this patch is used by client code (this patch itself doesn't

initiate any messages), it will emit telemetry packets consisting of the tag TTAG_USER. The format of these packets consist of the standard telemetry header, followed by 1 32-bit word containing a zero, followed by the number of data words indicated by the packet length. If the clients of the patch issue "printf" calls, the data will consist of a single null-terminated ascii string.

Word 0: SYNC (0x736f4166)
Word 1: [0..9] Length (3 + "n"/4)
Word 1: [10..31] TTAG_USER
Word 2: 0
Word 3..Length: Data

Science Impact:

Since this patch "plays" with the hardware and telemetry software, the use of this patch may interfere with the smooth operation of science runs.

=====
Patch Name: compressall

Part Number: 36-58030.27
Version: A
SCO: 36-1027
Environment: flight

Conflicts:
Depends On:
Size: 2368 bytes

Bcmd File: opt_compressall.bcnd
Pkts File: opt_compressall.pkts

Description:

This patch ensures that all raw mode packets are written to the telemetry stream without data loss. It eliminates the prior behavior in which, if a compressed pixel row was too long to fit into an output packet, the entire row was skipped and a zero-data-length was telemetered.

In the new version, rows that are too long when compressed are written uncompressed, with the telemetry packet header fields rewritten to indicate that that particular packet is uncompressed.

Applicable Reports/Requests:
 SPR-134

 SER-none

Test Results:
 reproduce --> PASS
 fix --> PASS

Replaced Functions:
 PmCcRaw::digestRawRecord
 PmTeRaw::digestRawRecord

Command Impact:
 None.

Telemetry Impact:
 Ground software must examine the compressionTableSlotIndex and compressionTableIdentifier fields of all dataCcRaw and dataTeRaw packets. If their values are 255 and 0, respectively, the pixel array should not be decompressed.

Science Impact:
 None. Raw mode is intended for diagnostic purposes only.

=====
Patch Name: ccignore

Part Number: 36-58030.10
Version: A
SCO: 36-1004
Environment: flight

Conflicts:
Depends On:
Size: 36 bytes

Bcmd File: opt_ccignore.bcnd
Pkts File: opt_ccignore.pkts

Description:
This patch causes the FEP to ignore "ignoreInitialFrames"
frames of data at the onset of Continuous Clocking data processing.

Applicable Reports/Requests:
SER-PENDING

Test Results:
smoke --> PASS

Replaced Functions:

Command Impact:
This patch will cause the start up time of a Continuous
Clocking run to increase by "ignoreInitialFrames" times
the frame rate configured for the run. If "ignoreInitialFrames"
is less than 2, the 2 frames will be skipped.

Telemetry Impact:
When "ignoreInitialFrames" is greater than 2,
the first telemetered Continuous Clocking exposure number
will be "ignoreInitialFrames", rather than "2".

Science Impact:
This may reduce the amount of noise in the early
telemetered frames of the Continuous Clocking run by
running the CCDs longer before processing and sending the data.

=====
Patch Name: eventhist

Part Number: 36-58030.05
Version: B
SCO: 36-1025
Environment: flight

Conflicts:
Depends On: smtimedlookup
Size: 5908 bytes

Bcmd File: opt_eventhist.bcnd
Pkts File: opt_eventhist.pkts

Description:

This patch implements the Event Histogram Mode. In this mode, the instrument performs the standard timed exposure clocking, and event detection and filtering, but rather than send the events to telemetry, the instrument builds CCD quadrant specific histograms of the summed corrected pulse heights of the accepted events. These histograms contain bins 0 through 4095. Events with a pulse height above 4095 are counted in bin 4095 and events with a negative value are counted in bin 0. All histogram bin values consist of a 26-bit count, followed by 5-bit of Hamming error detection/correction code, and 1 spare bit. The code is capable of detecting and correcting 1-bit errors in the count and hamming code bits.

Important: This version of the eventhist patch will only run correctly if the smtimedlookup patch is also loaded.

Applicable Reports/Requests:

Test Results:

smoke --> PASS
smoke2 --> PASS

Replaced Functions:

smTimedLookup3x3[3]
smTimedLookup5x5[3]

Command Impact:

As in normal Raw Histogram Mode, Event Histogram mode can only be used for Timed Exposure Science runs, and not in Continuous Clocking runs.

This mode is invoked by using the FEP_TE_MODE_EV3x3 or FEP_TE_MODE_EV5x5 for the fepMode field of the Timed Exposure Parameter Block, in conjunction with the new BEP_TE_MODE_EVHIST (3) for the bepPackingMode field.

Refer to the ACIS Software IP&CL Structure Definitions, Rev. M for details.

Telemetry Impact:

This mode defines new telemetry formats, TTAG_SCI_TE_REC_EV_HIST for exposure records, and TTAG_SCI_TE_DAT_EV_HIST for histogram data

packets. This new mode now places the count of error corrections performed on the quadrant's histogram bins within the previously unused "Variance Overclock High" of the exposure record, TTAG_SCI_TE_REC_EV_HIST. The Rev. M version of IP&CL renames this field accordingly.

The size of these packets are the same as those for TTAG_SCI_TE_REC_HIST and TTAG_SCI_TE_DAT_HIST respectively.

This mode always requires 10 telemetry buffers for each quadrant it accumulates (9 data buffers + 1 exposure record buffer per histogram). When accumulating histograms from all 4 quadrants on all 6 CCDs, the system requires 216 data buffers, and once the histograms are complete, it requires an additional 24 exposure record buffers. ACIS is configured for 400 science telemetry buffers, and as such, has enough buffering to accumulate only 1 complete set of histograms at a time. This will cause time gaps between sets of histograms when no events are accumulated. These gaps will consist of complete exposures, so partial exposures will not be accumulated in the histograms. As the previous buffers are telemetered and released back to the telemetry pool, eventually enough buffers (to be exact, 56) will be available to hold the 2nd set of histograms. At 24Kbps (format 2), this results in a time gap on the order of half a minute to a minute, and, at 500bps (format 1), a gap on the order of a half an hour to 45 minutes.

The total transmission time for a set of histograms at 24Kbps is about 3 minutes, whereas at 500bps, it starts approaching 2 hours.

If only 5 CCDs are used, ACIS can double-buffer the histograms, eliminating this gap, assuming that the histogram count times the frame time (exposure time + overhead) is large enough to accommodate the transmission time of the histograms. The total transmission time for 5 CCDs at 24Kbps is about 2 minutes, and at 500bps, the transmission time approaches 1.5 hours.

Details of these formats are described in the ACIS Software IP&CL Structure Definitions, Rev. M.

Science Impact:

This mode produces a new type of data product, histograms of the corrected and summed pulse heights from filtered events.

=====
Patch Name: printswhouse

Part Number: 36-58030.08
Version: 01
SCO: 36-986
Environment: flight

Conflicts:
Depends On: tlmio
Size: 7240 bytes

Bcmd File: opt_printswhouse.bcnd
Pkts File: opt_printswhouse.pkts

Description:
This patch provides a diagnostic which prints software housekeeping reports to telemetry in real-time, using the tlmio package.

Applicable Reports/Requests:
TOOL-PENDING

Test Results:
No Tests Specified

Replaced Functions:
SwHousekeeper::report

Command Impact:
None

Telemetry Impact:
This patch will cause the system to emit TTAG_USER packets containing a null terminated string, which describes the software housekeeping element currently being reported. See a description of the tlmio patch, MIT 36-58030.07.

Science Impact:
See the tlmio patch, 36-58030.07

=====
Patch Name: dearepl

Part Number: 36-58030.12
Version: 02
SCO: 36-1010
Environment: engineering

Conflicts: deaeng
Depends On:
Size: 556 bytes

Bcmd File: opt_dearepl.bcnd
Pkts File: opt_dearepl.pkts

Description:

This patch provides the basic capability to fake the existence of a DEA. This patch is used when no DEA box is available, or one wants to test without actually talking to the DEA.

Applicable Reports/Requests:
TOOL-PENDING

Test Results:
No Tests Specified

Replaced Functions:

DeaDevice::sendCmd
DeaManager::writeData
DeaManager::checkLoads
DeaDevice::isReplyReady
DeaCcdController::updateRegister
DeaDevice::readReply
DeaDevice::isCmdPortReady

Command Impact:

This "fakes" the existence of the DEAs. Commands which read and write PRAM, SRAM or DEA hardware will not crash, but won't work either.

Telemetry Impact:

This will produce true fiction from the DEAs.

Science Impact:

Can't do any, since the patch replaces the interface to the real DEAs.

=====
Patch Name: teignore

Part Number: 36-58030.09
Version: A
SCO: 36-1003
Environment: flight

Conflicts:
Depends On:
Size: 36 bytes

Bcmd File: opt_teignore.bcnd
Pkts File: opt_teignore.pkts

Description:
This patch causes the FEP to ignore "ignoreInitialFrames"
frames of data at the onset of Timed Exposure data processing.

Applicable Reports/Requests:
SER-PENDING

Test Results:
smoke --> PASS

Replaced Functions:

Command Impact:
This patch will cause the start up time of a Timed Exposure
run to increase by "ignoreInitialFrames" times the frame
rate configured for the run. If "ignoreInitialFrames"
is less than 2, the 2 frames will be skipped.

Telemetry Impact:
When "ignoreInitialFrames" is greater than 2,
the first telemetered exposure number will be
"ignoreInitialFrames", rather than "2".

Science Impact:
This may reduce the amount of noise in the early
telemetered frames of the Timed Exposure run by running
the CCDs longer before processing and sending the data.

=====
Patch Name: smtimedlookup

Part Number: 36-58030.24
Version: A
SCO: 36-1025
Environment: flight

Conflicts:
Depends On:
Size: 3712 bytes

Bcmd File: opt_smtimedlookup.bcnd
Pkts File: opt_smtimedlookup.pkts

Description:

This patch replaces several "switch" statements in SmTimedExposure class methods with a set of lookup tables indexed by the value of the BepMode and FepMode fields from the current TE parameter block. If a table slot is empty, the corresponding mode will be treated as unimplemented. With this patch, it is therefore possible to add more than one new TE mode via optional patches without the need to deliver a version of each patch for every possible combination of the other patches. The following methods, tables, and indices are used:

Method	lookup table	index
SmTimedExposure::setupProcess	smTimedLookupMode smTimedLookup3x3 smTimedLookup5x5	FepMode BepPackingMode BepPackingMode
SmTimedExposure::setupFepBlock	smTimedSetupFep	FepMode
SmTimedExposure::terminate	smTimedTerminate	FepMode

These tables may be patched by an extension of the "func" directive in the *.pkg file used to describe an ACIS patch. Hence, the line

```
func smTimedLookupMode[4] Test2_SmTimedExposure::setupCtil
```

instructs the linker to insert the address of the setupCtil() method of the Test2_SmTimedExposure class into slot 4 of the smTimedLookupMode table, so that setupCtil() will be called when FepMode == 4.

Applicable Reports/Requests:

Test Results:
smoke --> PASS

Replaced Functions:
SmTimedExposure::terminate
SmTimedExposure::setupProcess
SmTimedExposure::setupFepBlock

Command Impact:

None.

Telemetry Impact:

None.

Science Impact:

None.

=====
Patch Name: ctireport1

Part Number: 36-58030.25
Version: A
SCO: 36-1026
Environment: flight

Conflicts:
Depends On: smtimedlookup
Size: 5452 bytes

Bcmd File: opt_ctireport1.bcnd
Pkts File: opt_ctireport1.pkts

Description:

This patch implements a variant of timed-exposure 3x3 faint event mode in which the presence of precursor charge in each of the three columns that can contribute to each event is encoded in the 16 "outlying" pixels of Te5x5 mode.

FEP patches are loaded after the default code by two additional calls to `fehManager.loadRunProgram` from `Test2_SmTimedExposure::setupCtilFep`. Once loaded, the FEPs are marked as having been reset, thereby causing the following run to reload their default code.

Within the FEP, additional stack space is reserved for the `ctilstk` structure that holds the row indices and bias-subtracted pixel values of the most recently located precursor charge in each CCD column.

The new `FEPtestCtil` routine is called from an inline patch within `FEPsciTimedEvent` in advance of the `FEPtestOddPixel` or `FEPtestEvenPixel` routines. When a threshold crossing is detected, `FEPtestCtil` clears the `ctilstk` array (if this is a new frame), calls `FEPtestOddPixel` or `FEPtestEvenPixel`, and then pushes the pixel value and row index onto `ctilstk`. If `ctilstk` is full, the most distant (by row) value is dropped.

`FEPappendCtil` is called by the patched FEP code in place of the original `FEPappend5x5` routine. It determines the maximum bias-subtracted pixel value in each column, then inspects the `ctilstk` stacks for those columns, and packs up to 15 precursor charge values (adu and row) into elements 1 through 15 of the `pe[]` array:

```
pe[i] = STORE_PIX(pixel - bias - delta_overshoot, row_index)
```

`pe[0]` contains three 4-bit fields, the number of successive `pe[]` precursor values corresponding to `col-1`, `col`, and `col+1` of the event.

Applicable Reports/Requests:

Test Results:
smoke --> PASS

Replaced Functions:
smTimedLookupMode[4]

```
smTimedTerminate[4]
smTimedSetupFep[4]
```

Command Impact:

This patch requires that the `smtimedlookup` patch must also be loaded. Once loaded, it is invoked by setting `fepMode = FEP_TE_MODE_CTII1` in a `loadTeBlock` packet, writing that packet to a parameter block slot, and then starting a timed-exposure science run from that slot. The uplink format is defined in the ACIS IP&CL document 36-53204.0204 Rev. N.

Telemetry Impact:

The downlinked exposure and event data packets are identical in format to `exposureTeFaint` and `dataTeVeryFaint` except that their `formatTag` fields contain `TTAG_SCI_TE_REC_CTII1` and `TTAG_SCI_TE_DAT_CTII1`, respectively. When a `TTAG_SCI_TE_DAT_CTII1` is received, precursor charge data will be located in the `dataTeVeryFaint.pulseHeights` array, as follows:

```
pulseHeights[0]           - three 4-bit counters
pulseHeights[1..5,9,10,14,15,19..24] - precursor ADU and row
```

The sub-fields of `pulseHeights[0]` determine the contents of the other 15 fields:

```
ncol[0] = (pulseHeights[0] >> 8) & 15 -
ncol[1] = (pulseHeights[0] >> 4) & 15 -
ncol[2] = pulseHeights & 15           -
```

The fields from `icol-1`, if any, are written starting at `pulseHeights[1]`, followed by those from `icol`, and finally those from `icol+1`. The ADU values are stored in the 7 most significant bits of `pulseHeights[]` and the row indices in the least significant 5 bits, and should be extracted as follows:

```
adu = pulseHeights[i] & 0xfe0;
row = (pulseHeights[i] & 0x01f) << 5;
```

Unused `pulseHeights[]` will be filled with zeroes.

Science Impact:

This patch is intended for on-orbit diagnostic use only.

=====
Patch Name: txings

Part Number: 36-58030.33
Version: A
SCO: none
Environment: flight

Conflicts:
Depends On:
Size: 3128 bytes

Bcmd File: opt_txings.bcnd
Pkts File: opt_txings.pkts

Description:

With the continuing degradation of Chandra's EPHIN radiation monitor, an alternative is needed to permit the observatory to take the actions necessary to preserve its instruments during times of high solar activity. A recent analysis [Grant et al., 2010] has shown that, in some circumstances, the signature of solar events can be detected within the counts of CCD threshold crossings that are included in downlinked telemetry.

The txings patch monitors threshold crossings and uses ACIS bi-levels to communicate an alarm to the Chandra On-Board Computer (OBC). Event records are read from the FEP-BEP ring buffers by the processRecord() methods of the PmEvent, PmHist, and PmRaw classes. Each calls EventExposure::copyExpEnd() to parse the FEPexpEndRec records that contain thresholds, the count of threshold crossings, and expnum, the exposure number, but this routine doesn't have access to the ccdId that labels the record and which is needed to accumulate the crossings from that particular CCD.

The MIPS CPU architecture makes it relatively easy to make inline patches that permit additional arguments to be passed to subroutines. In the current case, we patch the routines that call copyExpEnd() in order to pass an extra argument. When processRecord() is called with a PmEvent object, this argument will be the address of the object, but for other callers, i.e., PmHist or PmRaw, the argument will be null to show that these modes don't count threshold crossings. Since PmEvent is a subclass of ProcessMode, the ccdId value can then be determined by a call to getCcdId(). A replacement for copyExpEnd() is called with an object of class EventExposure, and it calls saveTXings() with a static TXings object named txings in which the threshold crossing accumulators are stored.

The saveTXings() method is called once for each event-mode exposure frame. The first time that it is called in a science run, it determines the number of read-out rows, the maximum anticipated number of non-pathological threshold crossings per frame, and the frame exposure time in units of the FEP pixel clock (i.e., 10 us), and it increments the tx.threshold_accum and tx.exposure_accum accumulators. Integration times of less than 2000 seconds are guaranteed not to overflow either accumulator. Since the number of rows per frame and the frame exposure time are constant in continuous clocking mode, they are initialized in the TX structure, but in timed-exposure mode, the frame time depends on the dutyCycle, primaryExposure, and secondaryExposure parameters. These are extracted from the external pramTe object, where they were copied from the science run parameter block when the run started.

The radiation triggering algorithm is run in the triggerRadmon() routine. It is called every 64 seconds whether or not a science run is in progress. If it isn't, tx.count is set to zero until a subsequent call to saveTXings() from copyExpEnd() reloads the TX parameter structure from TXnext.

After the TXings patch has been uploaded and the BEP warm-booted, the tx.count field will be initialized to zero by the patch loader. The first time an event-mode science run reads a FEPexpEndRec record from the FEP-BEP ring buffer, it will call saveTXings(), which will reinitialize the radiation filter parameters from the TXnext structure. This makes it easy to change the filter parameters for subsequent science runs. When a trigger occurs, triggerRadmon() sets tx.triggered to BoolTrue and commands the memory manager thread to send a bepReadReply packet to telemetry, reporting the values of the txings parameters and variables. Then Test_Leds::show() sets the software bi-level channels to LED_BOOT_SPARE1, which persists for the remainder of the science run. After the science run ends, the next call to Leds::show() calls triggerRadmon() which sets tx.count to zero and tx.triggered to BoolFalse, canceling the special bilevel value and preventing threshold crossing triggers until the next science task starts, calls saveTXings(), and reloads the TX structure.

Once it is included in a patch load, and the BEP is warm-booted, the txings patch will be active during all subsequent science runs. When triggered by high and increasing threshold crossings, it sets the ACIS software bilevel values to LED_BOOT_SPARE1 until the science run ends, or until the tx.triggered field is explicitly cleared by a writeBep command. This guarantees that it will appear in Chandra major frame readouts (once per 32.4 seconds). The OBC should be patched to examine the ACIS bi-levels. It should save the instruments if (a) RADMON is enabled, and (b) the bi-level channels (1STAT3ST-1STAT0ST) have the LED_BOOT_SPARE1 values (1, 1, 0, 1).

Applicable Reports/Requests:

Test Results:

smoke --> PASS

Replaced Functions:

EventExposure::copyExpEnd
Leds::show

Command Impact:

The default filter parameters can be overridden by sending single writeBep command to ACIS to change the contents of the TXinit structure, whose address will depend on the ACIS flight software patch level (e.g., 0x8003dc30 in the current level E-F-G version). The command

```
write 0 0x8003dc30 {  
  0  
}
```

will, for instance, suspend the threshold crossing filter, and

```
write 0 0x8003dc30 {  
  5  
}
```

will turn it on again with an integration time of 5 minutes.

After a trigger, the bi-levels are not reset until `Leds::show()` is called when a science run is not in process. In the unlikely event that there is less than 64 seconds between the end of the triggering run and the start of the next, the bi-levels will continue to report `LED_BOOT_SPARE1`. This can be prevented by issuing a `writeBep` command to clear the counters:

```
write 0 0x8003dc90 {
    0 0
}
```

prior to the second `startScience`.

In normal operation, most science runs can be conducted with `txings` enabled, but exceptionally bright targets observed by few CCDs may lead to false triggers. It might be best to disable `txings` for short runs where the risk of radiation damage is small, or turn on additional CCDs for longer runs to reduce the likelihood of a false trigger. To change the trigger parameters for the next science run only, a `writeBep` command should update the fields in `TXnext` rather than `TXinit`, and this must be done before the science run has started to report events. In the current level E-F-G version, `TXnext` is located at `0x8003dc50`.

Telemetry Impact:

When a threshold crossing trigger occurs, `triggerRadmon()` commands the BEPs memory manager to write a `bepReadReply` packet to telemetry, reporting the contents of the `TX` and `tx` structures. If this action is blocked for any reason, a `SWSTAT_CMDECHO_DROPPED` event will be reported in software housekeeping.

The current version of the patch reports `bepReadReply` packets with a `formatTag` of `TTAG_READ_BEP`. If this causes confusion, a new `TlmFormatTag` value could be defined, but the CXC Data System would need to be reconfigured to handle it. Similarly, if `SWSTAT_CMDECHO_DROPPED` is confusing, a new `SwStatistic` value could be defined.

Science Impact:

None

=====
Patch Name: ctireport2

Part Number: 36-58030.26
Version: A
SCO: 36-1026
Environment: flight

Conflicts:
Depends On: smtimedlookup
Size: 2784 bytes

Bcmd File: opt_ctireport2.bcnd
Pkts File: opt_ctireport2.pkts

Description:

This patch implements a variant of timed-exposure 3x3 faint event mode in which the presence of precursor charge in each of the three columns that can contribute to each event is encoded in the low-order bits of three of the corner pixels.

FEP patches are loaded after the default code by two additional calls to `fepManager.loadRunProgram` from `Test3_SmTimedExposure::setupCtilFep`. Once loaded, the FEPs are marked as having been reset, thereby causing the following run to reload their default code.

Within the FEP, additional stack space is reserved for the `cti2stk` structure that holds the row indices of the most recently located precursor charge in each CCD column.

The new `FEPtestCti2` routine is called from an inline patch within `FEPsciTimedEvent` in advance of the `FEPtestOddPixel` or `FEPtestEvenPixel` routines. When a threshold crossing is detected, `FEPtestCti2` clears the `cti2stk` array (if this is a new frame), calls `FEPtestOddPixel` or `FEPtestEvenPixel`, and then updates `cti2stk` to indicate that this column contains charge.

`FEPappendCti2` is called by the patched FEP code instead of the original `FEPappend5x5`. It finds the maximum of the 4 corner pixels of the event that is being reported. Then it determines whether any of the three contributing columns contained precursor charge. Finally, it encodes this information in the low order bytes of the three smallest corner pixels. (Since the low-order bit of each corner pixel may be replaced, only the 11 high-order bits are compared when determining the maximum value).

Applicable Reports/Requests:

Test Results:
smoke --> PASS

Replaced Functions:
smTimedLookupMode[5]
smTimedTerminate[5]
smTimedSetupFep[5]

Command Impact:

The uplink format is defined in the ACIS IP&CL document 36-53204.0204 Rev. N. The `fepMode` field in the `loadTeBlock` command packet must be set equal to `FEP_TE_MODE_CTI2`. Unless the `smtimedlookup` patch has also been loaded, this value will cause a subsequent `startScience` command that references this parameter block to fail.

Telemetry Impact:

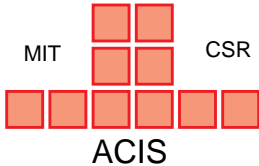
The downlinked exposure and event data packets are identical in format to `exposureTeFaint` and `dataTeFaint`. To process the precursor charge information, ground software must first inspect the `loadTeBlock` reported in the `dumpedTeBlock` packet that started the run. If the `fepMode` field is equal to `FEP_TE_MODE_CTI2`, subsequent `dataTeFaint` packets should be inspected. The following code fills `ee[i]` with one (zero) according to whether column (`ccdColumn+i-1`) did (did not) contain precursor charge:

```
unsigned nn, mm, ii, ee[3];

for (mm = 0, nn = 2; nn < 9; nn++) {
    if ((nn & 1) == 0 && nn != 4) {
        if ((pulseHeights[nn] & 0xffe) > (pulseHeights[mm] & 0xffe))
            mm = nn;
    }
}
for (nn = ii = 0; nn < 9; nn++) {
    if ((nn & 1) == 0 && nn != 4 && nn != mm) {
        ee[ii++] = pulseHeights[nn] & 1;
    }
}
```

Science Impact:

This patch is intended for on-orbit diagnostic use only.

		ENGINEERING CHANGE ORDER		<u>ECO No.</u> <u>36-1046</u>	
CENTER FOR SPACE RESEARCH MASSACHUSETTS INSTITUTE OF TECHNOLOGY					
DWG. NO.		NEW REV.	DRAWING TITLE		
36-58021.04		G	Flight Software Patch Release E-F-G Certification		
REASON FOR CHANGE: Certification of standard patch release E with optional patch release F which includes the <i>txings</i> patch, along with the same optional patches that were certified in release E-E-F, <i>i.e.</i> , <i>smtimedlookup</i> , <i>compressall</i> , <i>eventhist</i> , <i>cc3x3</i> , and <i>untricklebias</i> .					
DESCRIPTION OF CHANGE: Three optional patch combinations are certified as release E-F-G: (a) <i>smtimedlookup</i> , <i>eventhist</i> , <i>cc3x3</i> , <i>compressall</i> and <i>txings</i> . (b) <i>smtimedlookup</i> , <i>eventhist</i> , <i>cc3x3</i> , <i>compressall</i> , <i>untricklebias</i> and <i>txings</i> . The certification tests are taken from these specific combinations of the optional release F patches, with the full set of standard patches, release E.					
		SIGNATURE	DATE	REMARKS:	
ORIGINATOR		RFG	03/02/11	Reviewed and signed-off.	
MECHANICAL					
ELECTRICAL					
SOFTWARE					
STRUCTURE					
FABRICATION					
SCIENCE					
SYSTEMS ENG.					
QUALITY					
PROJ. ENGINEER					
DEPUTY PM					
PROJ. MANAGER					

07/28/11
18:52:17

Flight S/W Patches, Revision E-F-G

1

../../certsrc/cc3x3+eventhist+compressall+txings.notes

TITLE: ACIS eventhist, cc3x3, txings, compressall, smtmedlookup Patch Certification Release Notes

DOCUMENT NUMBER: 36-58021.03 REVISION: G

ORIGINATOR: Peter G. Ford <pgf@space.mit.edu>

LETTER	SCO NO.	DESCRIPTION	APPROVED	DATE
G	36-1046	Certify Rev-E-Opt-F patches	RFG	03/02/2011
G	36-1046	Rev. E Standard and Rev. F. Opti		

=====
Title: ACIS eventhist, cc3x3, txings, compressall, smtimedlookup Patch Certification Release
Notes for Version G

Software Change Order: 36-1046

Build Date: Thu Jul 28 18:52:16 EDT 2011
Part Number: 36-58021.03
Version: G
CVS Tag: cc3x3+eventhist+compressall+txings-E-F-G

Std Number: 36-58010
Std Version: E
Std Tag: release-E
Std SCO: 36-1042

Opt Number: 36-58020
Opt Version: F
Opt Tag: release-E-opt-F
Opt SCO: 36-1044

IPCL Number: 36-53204.0204
IPCL Version: N
IPCL CVS Tag: release-N

Description:

This certification verifies the operation of the Continuous Clocking 3x3, Event Histogram, Compress All, Science Mode Timed Lookup, and Threshold Crossing Trigger Patches.

The certification consists of six tests, copied from the original test run during the Options Release. The tests have been modified to load all four optional patches, rather than just one at a time, and to clean up some false failures due to timing/pattern matching issues in the tests.

The tests verify that the patch modes run as they did during the original test when they are both installed into the system.

The Continuous Clocking 3x3 (cc3x3) test consists of two parts. The first launches a CC3x3 run, whereas the second runs CClx3. This suite performs CClx3 tests to verify that the modifications to the existing BEP Continuous Clocking functions do not break the existing CClx3 functionality. Since the FEP software only contains CC3x3 code during CC3x3 runs (this is verified by the CClx3 run), and no BEP functions used by Timed Exposure are modified by the patch, the Timed Exposure modes do not need to be re-tested as part of this certification.

Each test sends a series of events on the right side of each quadrant (the original test was derived from the test for the rquad bug fix), and verifies that the mode runs nominally, and produces the expected event list. Since the "stop" critereon for the test is a little fuzzy, the runs tend to produce additional exposures that aren't in the file used to check the run's event output. "diff" used in the test produces mismatches on the additional exposures produced by the test run. Manual check of the run data shows that the event lists are replicated correctly by the run. Later, a "wrapping" comparison may be developed to eliminate this manual step.

The Event Histogram test uses a similar strategy to the CC3x3 test. It starts an Event Histogram run, and sends in a series of standard

events. It then compares the resulting quadrant histograms with an example file to verify the results.

One caveat that arose during the review of the Optional patches is that, when the standard patch "zaplexpo" is present, which it should always be, the first exposure of event histogram mode will not contain any events. This will cause the first histogram from each FEP quadrant to appear to have been integrated for 1 less frame time than subsequent quadrant histograms. This is different than Raw Histogram mode, which is not affected by the "zaplexpo" patch. The histogram example file used for this certification assumes that no events are sent during exposure 2 (the first "real" exposure of the run).

The smTimedExposure patch is tested by merely running a timed-exposure faint run, verifying that the bias and event detection phases have been invoked, and then stopping the run.

The Compress All patch is tested by copying an image to the image loader that contains several very "noisy" rows that are known to be incompressible by the Huffman tables. A timed-exposure raw-mode run is executed and the pixelCount field of the dataTeRaw packets of a couple of raw frames is monitored. The test fails if pixelCount is ever zero.

The Threshold Crossing Trigger patch, txings, conducts a series of science runs -- timed exposure 3x3, event histogram, and raw, and continuously clocked 3x3, 1x3, and raw, increasing the threshold crossing rate and monitoring the ACIS bi-levels for the trigger signal, accompanied by the appropriate bepReadReply packet.

Included Patches:

eventhist
cc3x3
txings
compressall
smtimedlookup

Test Support Patches:

printswhouse
dearepl
tlmio

Test Results:

smtimedlookup --> PASS
cc3x3 --> PASS
eventhist --> PASS
eventhist --> PASS
compressall --> PASS
txings --> PASS

07/29/11
00:15:28

Flight S/W Patches, Revision E-F-G

1

../../../../certsrc/cc3x3+eventhist+compressall+untricklebias+txings.notes

TITLE: ACIS untricklebias, eventhist, cc3x3, txings, compressall, smtimedlookup Patch Certification Release Notes

DOCUMENT NUMBER: 36-58021.03 REVISION: G

ORIGINATOR: Peter G. Ford <pgf@space.mit.edu>

LETTER	SCO NO.	DESCRIPTION	APPROVED	DATE
G	36-1046	Certify Rev-E-Opt-F patches	RFG	03/02/2011
G	36-1046	Rev. E Standard and Rev. F Optio		

=====
Title: ACIS untricklebias, eventhist, cc3x3, txings, compressall, smtimedlookup Patch Certification Release Notes for Version G

Software Change Order: 36-1046

Build Date: Fri Jul 29 00:15:27 EDT 2011
Part Number: 36-58021.03
Version: G
CVS Tag: cc3x3+eventhist+compressall+untricklebias-txings-E-F-G

Std Number: 36-58010
Std Version: E
Std Tag: release-E
Std SCO: 36-1042

Opt Number: 36-58020
Opt Version: F
Opt Tag: release-E-opt-F
Opt SCO: 36-1044

IPCL Number: 36-53204.0204
IPCL Version: N
IPCL CVS Tag: release-N

Description:

This certification verifies the operation of the Continuous Clocking 3x3, Event Histogram, Compress All, Untrickle Bias, Science Mode Timed Lookup, and Threshold Crossing Trigger Patches.

The certification consists of seven tests, copied from the original test run during the Options Release. The tests have been modified to load all four optional patches, rather than just one at a time, and to clean up some false failures due to timing/pattern matching issues in the tests.

The tests verify that the patch modes run as they did during the original test when they are both installed into the system.

The Continuous Clocking 3x3 (cc3x3) test consists of two parts. The first launches a CC3x3 run, whereas the second runs CClx3. This suite performs CClx3 tests to verify that the modifications to the existing BEP Continuous Clocking functions do not break the existing CClx3 functionality. Since the FEP software only contains CC3x3 code during CC3x3 runs (this is verified by the CClx3 run), and no BEP functions used by Timed Exposure are modified by the patch, the Timed Exposure modes do not need to be re-tested as part of this certification.

Each test sends a series of events on the right side of each quadrant (the original test was derived from the test for the rquad bug fix), and verifies that the mode runs nominally, and produces the expected event list. Since the "stop" critereon for the test is a little fuzzy, the runs tend to produce additional exposures that aren't in the file used to check the run's event output. "diff" used in the test produces mismatches on the additional exposures produced by the test run. Manual check of the run data shows that the event lists are replicated correctly by the run. Later, a "wrapping" comparison may be developed to eliminate this manual step.

The Event Histogram test uses a similar strategy to the CC3x3 test. It starts an Event Histogram run, and sends in a series of standard

```
../../certsrc/cc3x3+eventhist+compressall+untricklebias+txings.notes
```

events. It then compares the resulting quadrant histograms with an example file to verify the results.

One caveat that arose during the review of the Optional patches is that, when the standard patch "zaplexpo" is present, which it should always be, the first exposure of event histogram mode will not contain any events. This will cause the first histogram from each FEP quadrant to appear to have been integrated for 1 less frame time than subsequent quadrant histograms. This is different than Raw Histogram mode, which is not affected by the "zaplexpo" patch. The histogram example file used for this certification assumes that no events are sent during exposure 2 (the first "real" exposure of the run).

The smTimedExposure patch is tested by merely running a timed-exposure faint run, verifying that the bias and event detection phases have been invoked, and then stopping the run.

The Compress All patch is tested by copying an image to the image loader that contains several very "noisy" rows that are known to be incompressible by the Huffman tables. A timed-exposure raw-mode run is executed and the pixelCount field of the dataTeRaw packets of a couple of raw frames is monitored. The test fails if pixelCount is ever zero.

The Untrickle Bias patch is tested by a pair of expect scripts, each of which performs 12 tests, one in TE mode, the other in CC mode. Each test starts a science run and then terminates it in one of the possible ways, viz:

- 1: stopScience during bias map creation
- 2: double stopScience during bias map creation
- 3: startScience during bias map creation
- 4: assert/deassert RADMON during bias map creation
- 5: stopScience during bias map telemetering
- 6: double stopScience during bias map telemetering
- 7: startScience during bias map telemetering
- 8: assert/deassert RADMON during bias map telemetering
- 9: stopScience during event processing
- 10: double stopScience during event processing
- 11: startScience during event processing
- 12: assert/deassert RADMON during event processing

The tests fail unless all steps complete and return the anticipated scienceReport return codes.

The Threshold Crossing Trigger patch, txings, conducts a series of science runs -- timed exposure 3x3, event histogram, and raw, and continuously clocked 3x3, 1x3, and raw, increasing the threshold crossing rate and monitoring the ACIS bi-levels for the trigger signal, accompanied by the appropriate bepReadReply packet.

Included Patches:

```
untricklebias
eventhist
cc3x3
txings
compressall
smtimedlookup
```

Test Support Patches:

```
printswhouse
dearepl
```

tlmio

Test Results:

```
smtimedlookup --> PASS
cc3x3 --> PASS
eventhist --> PASS
eventhist --> PASS
compressall --> PASS
untricklebias --> PASS
untricklebias --> PASS
txings --> PASS
```