r e p o r t

MIT Kavli Institute
1 Hampshire Street
Cambridge MA 02139–4307
Tel: 617-253-7277
Fax: 617-253-8084

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

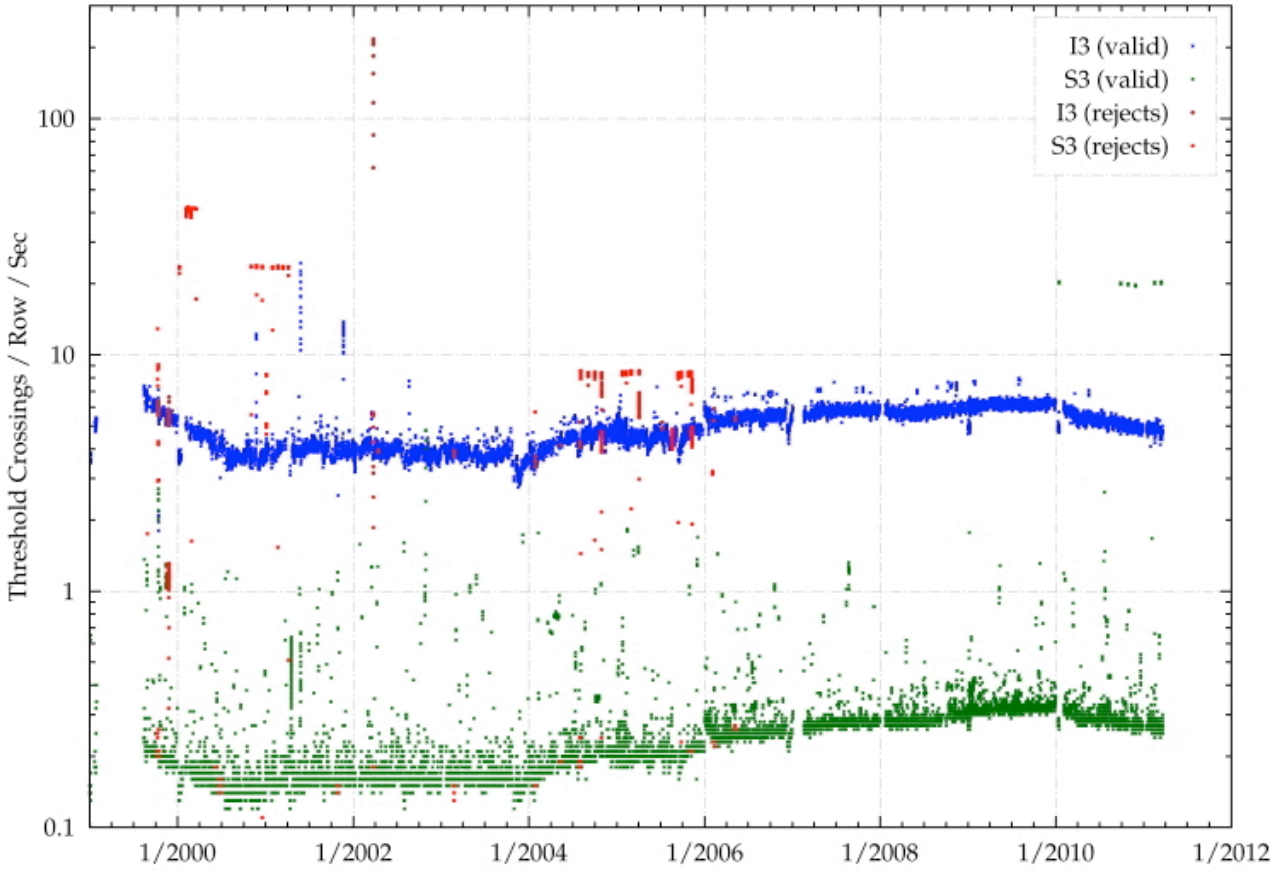
[1] In this report, names of command and telemetry packets and their fields are written in *italics*; the names of classes, methods, objects, and variables of the ACIS flight software are written in a `typewriter font`.

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**primaryExposure* seconds, followed by *dutyCycle* frames at 0.1**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.

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. Performing Science Run.
LED_RUN_SCIENCE_B	x x x x 0 1 0 1	
LED_RUN_IDLE_A	x x x x 0 1 1 0	Most recent boot was commanded. Not performing science run.
LED_RUN_IDLE_B	x x x x 0 1 1 1	
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 μs), 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 `FEPparmBlock` 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 runtest.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 ~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 *fepthrottle* 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 "fepCtl.h"

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

The *fepthrottle* patch also includes inline patches to call `throttleFepAppendRingBuf()` in place of `fepAppendRingBuf()` from within the event-processing routines in *fepSciTimed.c* and *fepSciCClk.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 *fepthrottle* 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. (*q.v.*)
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. *Mongoose* 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 (*q.v.*)
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 *txings* 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.