I N T E G R A L

SPECTROMETER

SPI

SPI-MU-0-1062V3-CNES
Issue   : 5
Revision : 0
Date   : 28/02/02
Page No. : ANX9-1

ANNEX 9

PSD DESCRIPTION SOFTWARE

SPI-NT-4232-4194-CESR

Ed. 2   Rev. 1

PSD Software Description

Jürgen Knödlseder[*]

April 9, 2001

1   Scope

The scope of this document is to provide a description of the software that is implemented in the PSD FM subassembly of the INTEGRAL spacecraft. It describes the scientific analysis algorithm and explains its implementation in the form of a DSP assembler code. It also defines the interface between the scientific code and the functional code (called hereafter eng).

2   Software changes

This document describes the scientific software version 1.07. The following sections summarise the software changes with respect to precedent software versions.

2.1   Changes with respect to V1.06

- Conform to annex 21, only the three least significant Bits of the first of the four library selection parameters (Byte 3 for channel 0) are used for library set selection.

- The baseline running averages are copied in the corresponding memory blocks of the eng code (using the lib_status field. They are now accesible in the housekeeping telemetry.

- The pulse_threshold parameter was deleted from the library parameter block. In V1.06, the pulse peak height was required to exceed pulse_threshold in order to be analysed. This test was removed.

- A condition was added (replacing the pulse_threshold test) that verifies that the pulse area is comprised within a lower and a upper limit (minpulse and maxpulse, respectively). Corresponding boundary parameters were added to the library parameter block.

- The smaller pulse threshold parameter maxthres was split-up into two parameters maxthresneg and maxthrespos that allow for different discrimination parameters for negative and positive peak spacing. 10 new parameters were introduced in the library parameter block.

3   Formulation of the PSD algorithm

The PSD subassembly of the SPI telescope aboard the INTEGRAL spacecraft consists of an onboard characterisation of detector current pulses aiming in the reduction of instrumental background. The characterisation is done by a comparison of a measured pulse shape with a library of reference pulse shapes that have been determined during a calibration phase (either ground or inflight calibration). The comparison is based on a χ^2 goodness-of-fit test that determines the best fitting linear combination of two reference pulses under the constraint that the total area under the measured pulse equals the total area under the pair of reference pulses[1]. Further, the amplitude of both pulses is constrained to be positive,

---

[*]Centre d'Etude Spatiale des Rayonnements (CNRS/UPS), 9, avenue du Colonel Roche, B.P. 4346, 31028 Toulouse Cedex 4, FRANCE
[1]Tests with an unconstrained version of the goodness-of-fit test have shown no difference in the resulting PSD efficiency, hence the pulse area constraint can be considered as valid.

2   *PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001*

hence the relative amplitudes α and $1-\alpha$ of the first and the second reference pulse, respectively, lies in the interval $[0, 1]$.

In order to avoid testing all possible combinations of two reference pulses, a two step approach has been designed. In a first step, only a single reference pulse is fitted to the measured pulse in order to determine the first order best fitting template. In a second step, a pair of reference pulses is fitted to the measured pulse where the peak position (i.e. the time-to-peak or TTP value) of the primary pulse varies within ± 2 bins (i.e. ± 20 ns) around the best fitting single pulse peak position, while the peak position of the secondary pulse runs through all available reference pulses[2].

In the following, the mathematics of the fitting procedures is described and possible simplifications leading to an efficient coding are outlined.

3.1   Fast single pulse fitting

For a single reference pulse the χ^2 goodness-of-fit test reduces to the evaluation of

$$\chi^2 = \frac{1}{\sigma^2} \sum_i (c_i - f_i)^2 \qquad (1)$$

since both f_i and c_i are assumed to be normalised to unity. σ is the noise estimate in the observed pulse, but since we are only interested in the best-fitting template for a given measured pulse σ is constant and hence not of interest when searching the minimum of χ^2. Equation 1 can be further simplified by removing the constant leading term, resulting in a *modified* χ^2 *statistics*

$$\chi^2_{\text{mod}} = \sum_i f_i f_i - 2 f_i c_i. \qquad (2)$$

Note that only the term $f_i c_i$ depends of the measured pulse while $f_i f_i$ can be pre-calculated at initialisation.

3.2   Fast double pulse fitting

Fitting of a linear combination of two reference pulses f_i and g_i is done by minimisation of

$$\chi^2 = \frac{1}{\sigma^2} \sum_i (c_i - \alpha f_i - (1-\alpha) g_i)^2 \qquad (3)$$

where σ is the noise estimate, c_i is the measured pulse shape, and α is the relative pulse amplitude, called also the mixing parameter. Note that in this formulation it is assumed that the areas of the pulse shape c_i and the library templates f_i and g_i have been normalised to unity.

To minimise Eq. (3), the derivate is calculated and set to zero. The mixing parameter α that fulfils this constraint is then given by

$$\alpha = \frac{-\sum_i f_i g_i + \sum_i f_i c_i - \sum_i g_i c_i + \sum_i g_i g_i}{\sum_i f_i f_i + \sum_i g_i g_i - 2 \sum_i f_i g_i}. \qquad (4)$$

To find the best fitting pair (f_i, g_i) of templates the minimum χ^2 is searched. Again, the best fitting pair is determined by a relative comparison, hence constant terms can be omitted and the *modified* χ^2 *statistics*

$$\chi^2_{\text{mod}} = \sum_i \left[-2\alpha f_i c_i - 2(1-\alpha) g_i c_i + \alpha^2 f_i f_i + 2\alpha(1-\alpha) f_i g_i + (1-\alpha)^2 g_i g_i \right] \qquad (5)$$

may be used. Using the definitions

$$n = -\sum_i f_i g_i + \sum_i f_i c_i - \sum_i g_i c_i + \sum_i g_i g_i \qquad (6)$$

and

$$d = \sum_i f_i f_i + \sum_i g_i g_i - 2 \sum_i f_i g_i \qquad (7)$$

the minimisation procedure reduces to the following scheme:

---

[2] It has been demonstrated that searching ± 2 bins around the peak of the best fitting single pulse is sufficient and increasing of the search region does not lead to an improvement of the PSD efficiency.

PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001 3

- Evaluate n
- Evaluate $\alpha = n/d$
- Evaluate $\chi^2_{mod} = \sum_i \left[g_i^2 - 2 g_i c_i - \alpha n \right]$

The denominator d and $\sum_i g_i^2$ do not depend on c_i, hence may be pre-calculated at initialisation. The last equation can be understood by considering that

$$
\begin{aligned}
\chi^2_{mod} &= \sum_i \left[-2\alpha f_i c_i - 2 g_i c_i + 2\alpha g_i c_i + \alpha^2 f_i^2 + 2\alpha f_i g_i - 2\alpha^2 f_i g_i + g_i^2 - 2\alpha g_i^2 + \alpha^2 g_i^2 \right] \\
&= \sum_i \left[g_i^2 - 2 g_i c_i + 2\alpha \left(g_i c_i + f_i g_i - f_i c_i - g_i^2 \right) + \alpha^2 \left(f_i^2 - 2 f_i g_i + g_i^2 \right) \right] \\
&= \sum_i \left[g_i^2 - 2 g_i c_i - 2\alpha n + \alpha^2 d \right] \\
&= \sum_i \left[g_i^2 - 2 g_i c_i - 2\alpha n + \alpha n \right] \\
&= \sum_i \left[g_i^2 - 2 g_i c_i - \alpha n \right] \tag{8}
\end{aligned}
$$

Note that $\alpha = n/d$, i.e. $\alpha d = n$.

4 PSD software interface definition

The PSD onboard software consists of two major packages:

- the functional software (`eng`)
- the scientific software (`science`)

The functional software provides all code that handles the PSD interfaces (detector signals, telecommands and housekeeping via LSL, analysis telemetry via HSL, Veto signal, DFEE identifier and time-tag). It provides the main loop of the PSD subassembly, allows for the configurational control of the system, identifies the detector signals to be analysed, accumulates detector pulse shapes and schedules them for scientific analysis, gathers scientific analysis results and puts them into the telemetry, and collects statistics on the PSD performance.

The scientific software provides all code that analyses a measured pulse shape in order to discriminate single-site interactions from multiple-site events. Additionally, it handles the uploadable library templates and the writing of the EEPROMs. The interface between `eng` and `science` is done via four subroutine calls:

- `analinit` : initialises the scientific analysis package
- `anal` : scientific analysis routine called for each event
- `addlibrary` : adds a library template to memory
- `correlate` : pre-calculates library dependent constants

In the following, the interfaces between `eng` and the four scientific subroutines are defined in detail. Within the subroutines, all DSP32C registers except of `r14` and `r18` may be used. `r14` is a stack pointer that my be used by any code fragment for temporary parameter storage while `r18` holds the return address of the subroutine calls. During `addlibrary` and `correlate` errors may occur due to improperly specified parameters (out of boundary, etc.). These errors are signalled via an error code to the user (see section 9.3). Note, that multiple errors may occur during the execution of `correlate` where the actual error code that is accessible via the housekeeping command 0x13 is the last error that occurred.

4.1 analinit

This subroutine is called at power-up or reset (TBC) of the PSD system. It should perform actions that are only needed once during the code execution. Consequently, calculations done in this routine cannot depend on configurable parameters.

4.2 anal

This subroutine is called once for each measured pulse shape that should be analysed by the PSD subassembly. All information concerning the pulse shape is stored by eng in a structure of 256 Bytes length. The pulse shape itself is encoded as 96 16 bit integers, preceded by 4 16 bit words that contain

- the event identifier, including the detector number

- a null word (set to 0)

- the PSD information word, initially set to the PSD energy value

- the PSD energy value (used for energy discrimination by the eng software

A pointer stored in the variable rawevent points to the first 16 bit integer of the pulse shape (and NOT to the beginning of the 256 Byte structure).

The result of the scientific PSD analysis is handed back to the eng software via the PSD information word. Only the most significant bit (bit 15) of this word will be interpreted by the eng software (and all other software aboard of SPI/INTEGRAL) in order to update PSD selection statistics. The 15 remaining bits (bit 14 - bit 0) contain the result of the scientific analysis in a compressed format (see section 5.7.2) and will only be analysed on ground.

4.3 addlibrary

This subroutine is called after a complete set of library upload commands have been received by the PSD subassembly (configuration commands 0x0b-0x11). The data interface between eng and science is implemented via a data structure that consists of the following fields:

- libloaddetector : 16-bit word that holds the detector number

- libloadcurve : 16-bit word that holds the template curve number

- libloadset : 16-bit word that holds the library set number

- libloadlength : 16-bit word that holds the number of 32-bit words that should be uploaded

- libloaddata : 64 32-bit words to upload

Library data are floated by addlibrary and then copied to EEPROM via the EEPROM burning routine eewrite that is also part of the scientific software package.

Curve number 255 (or 0xff) is a special curve that holds PSD algorithm parameters as specified in section 7. These parameters are checked and also stored in EEPROM together with some derived parameters.

4.4 correlate

This subroutine is called at the following occasions:

- after initialisation of the system (after analinit)

- after a change of the library selection and control settings (any of the configuration commands 0x07, 0x08, 0x09)

This routine pre-calculates terms used for the scientific analysis that are not dependent of the measured pulses, such as normalised templates, inner products between and among templates, and the denominators used for fast double pulse fitting. These terms are calculated from the library templates in EEPROM and stored in RAM for fast memory access (hence EEPROM is not accessed during individual pulse analysis). Since each detector has its own library, each detector also has its own precalculated arrays.

5 PSD algorithm implementation

As illustrated in the above section, the calculations required for the pulse shape discrimination can be separated in pre-calculated terms and terms that have to be calculated for each pulse. This separation minimises the processing time required for each pulse but demands a considerable amount of memory for the storage of pre-calculated values. Pre-calculated terms depend only on the library templates and on the number of bins in a template. They are calculated during a call to **correlate**. The remaining terms that depend on the measured pulse are calculated during scientific analysis (call to **anal**).

The scientific PSD algorithm consists of six sections (labelled step 0-5):

0 : algorithm parameter initialisation

1 : pulse preparation

2 : pulse filtering

3 : calculation of inner products and single pulse fitting

4 : double pulse fitting

5 : pulse shape discrimination

All 6 steps are coded as separate subroutines and may be replaced inflight by memory uploads. In the following, these 6 steps are described in detail. Afterwards, the algorithm implementation of the remaining subroutines is given.

5.1 STEP 0 : Algorithm parameter initialisation

The following tasks are performed in the specified order:

5.1.1 Pulse parameter extraction

The basic parameters, i.e. the detector number and the pulse energy as determined by the PSD ADC, are extracted from the event data structure. The detector number of stored in the algorithm variable **detector**, the PSD ADC energy is sorted in **psd_energy**. The detector number is verified on validity (number between 0-18) and the pulse is rejected if an invalid detector number is found. Pulse analysis is only continued if a valid library is available for the requested detector. This is signalled by the corresponding Byte in the **lib_valid** array set to 1.

5.1.2 Base address determination

The base addresses of the pre-calculated arrays are determined for the requested detector and stored in variables for later use. This allows for fast pulse fitting and goodness-of-fit testing. The following base addresses are set:

- **libbase**: Normalised library templates in RAM
- **lxlbase**: Inner products among library templates
- **lxxbase**: Inner products between library templates
- **denbase**: Denominators for fast pulse fitting

In addition, the following detector dependent quantities are retrieved:

- **walpha**: The 15-bit weighting factor used to convert the second pulse amplitude into an integer value
- **addbaseavg**: The address of the baseline average
- **addbasenout**: The address of the number of baseline outliers

6   *PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001*

5.1.3 Read library control and parameter block

The algorithm parameters are extracted from the library control and parameter blocks. These parameters are detector dependent and, for the pulse shape discrimination parameters, also energy dependent. The following parameters are extracted from the library control block:

- n_temp_bins: Number of bins used for fitting of templates

- n_templates: Number of templates used for fitting

It has been tested in the `correlate` subroutine that these parameters fall in the valid range. Otherwise the library would be flagged as invalid.

The following parameters are extracted from the library parameter block:

- n_start_bins: Number of bins in 'start block'

- n_end_bins: Number of bins in 'end block'

- time_mid: Mean time for late/early peak separation

- pulse_dur_min: Minimum pulse duration

- pulse_dur_max: Maximum pulse duration

- pulse_saturate: Pulse saturation value

- inv_start_bins: 1/n_start_bins

- inv_end_bins: 1/n_end_bins

- thresh_frac: Threshold fraction

- base_this_fract: Baseline average fraction for actual baseline

- base_avg_fract: Baseline average fraction for baseline average

- base_outlier: Baseline average outlier value

- base_max_outlier: Maximum number of subsequent outliers

- minbase: Minimum baseline value

- maxbase: Maximum baseline value

- minpulse: Minimum pulse area value

- maxpulse: Maximum pulse area value

5.2 STEP 1 : Pulse preparation

The following tasks are performed in the specified order:

5.2.1 Pulse floating

The pulse is digitised with 9 Bit of resolution by 4 ADCs. The PSD engineering software provides 96 bins for each pulse to the scientific analysis routine. As first analysis step, these 96 Bits are converted to 96 floating point numbers (4 Byte each) and stored in RAM. All subsequent analysis is performed on the floated values.

5.2.2 ADC gain correction

Each of the 4 ADCs may be software gain corrected in order to compensate for differences in the hardware. For this purpose the floated pulse shape is transformed via

$$\text{pulse}_i = g_k \times \text{rawevents}_i + o_k \qquad (9)$$

where $k = i$ modulus 4, g_k is the gain of ADC k and o_k is the offset of ADC k. The gain correction and offset values are transferred to the PSD via the telemetry command 0x06 which contains a 8-bit field for each gain and each offset value. These fields can be accessed via the ad_offsets field of the engineering code and should be interpreted as 8-bit signed integers. They are converted to floating point values using

$$g_k = 1.0 + \text{GAIN\_STEP} \times gain_{8-bit} \qquad (10)$$

where GAIN_STEP = 0.0005 and

$$o_k = \text{OFFSET\_STEP} \times offset_{8-bit} \qquad (11)$$

where OFFSET_STEP = 0.05. (note that GAIN_STEP and OFFSET_STEP are specified at compile time). All subsequent analysis is performed on the gain corrected values.

5.2.3 Absolute time-to-peak

The absolute time-to-peak value is determined by searching the 96 bins of the pulse for the maximum. The resulting bin index is stored in the algorithm variable ATTP. The search is broken down in 3 search blocks, called 'start block', 'mid block', and 'end block'. Using this approach, the integral of the pulse in each of these blocks can be calculated in parallel. This integral will be used later for the determination of the baseline. The number if bins in the 'start block' and 'end block' are configurable in the library parameter block using the parameters n_start_bins and n_end_bins. The number of bins in the 'mid block' is given by

$$\text{N\_SHAPE\_BINS} - \text{n\_start\_bins} - \text{n\_end\_bins} \qquad (12)$$

where N_SHAPE_BINS is the number of bins in a pulse shape, usually 96 (specified at compile time). The pulse integral in the three blocks are stored in integral_start, integral_mid, and integral_end, the total integral is stored in the algorithm variable integral.

5.2.4 Rejection of saturated pulses

The maximum of the gain corrected pulse is compared to a pulse saturation value. If this value is exceeded, the pulse is rejected as saturated pulse. The pulse saturation value is determined as follows: the ADC number for which the maximum occurred is determined by taking the ATTP modulo 4, i.e. modulo the number of ADCs. An integer pulse saturation value, given by the variable pulse_saturate in the library parameter block (typically 510), is converted by the appropriate gain correction

$$s = g_k \times \text{pulse\_saturate} + o_k \qquad (13)$$

in order to obtain the saturation value in the gain corrected units (this makes pulse saturation independent of gain correction).

The pulse saturation value is transferred as 9-bit unsigned integer in the library parameter block.

5.2.5 Reject badly truncated pulses

Next, a test is performed on ATTP to reject truncated pulses. This eliminates pulses for which ATTP = 0 and ATTP = N_SHAPE_BINS − 1 (usually 95), i.e. the last bin in the pulse.

8   *PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001*

5.2.6 Baseline determination

The baseline for each pulse is determined from the integral of either the 'start block' or the 'end block', stored in `integral_start` and `integral_end`. If the pulse peak occurs early, i.e. ATTP $\leq$ `time_mid`, then the rear baseline is used for the analysis, calculated using

$$baseline = integral\_end \times inv\_end\_bins. \qquad (14)$$

Otherwise (i.e. for ATTP > `time_mid`) the front baseline is used for the analysis, calculated using

$$baseline = integral\_start \times inv\_start\_bins. \qquad (15)$$

In order to reduce the error in the baseline estimation, the baseline is averaged over a number of pulses using exponential averaging

$$baseline\_avg = baseline \times base\_this\_fract + baseline\_avg \times base\_avg\_fract, \qquad (16)$$

where `baseline_avg` is the average baseline of the last pulses for this detector, `base_this_fract` is the fraction of the actual baseline used for the average, and `base_avg_fract = 1 − base_this_fract` is its complement. The following scheme has been adopted to remove outliers from the baseline average: Any baseline for which

$$abs(baseline − baseline\_avg) > base\_outlier \qquad (17)$$

is considered as potential outlier. If such a pulse occurs, an outlier counter is incremented, and compared to the maximum number of allowed subsequent outliers `base_max_outlier` (typically 1-2). In case that the counter has exceeded the maximum it is highly probable that the 'outlier' is indeed a real variation of the baseline, hence it will be included in the baseline averaging. For such events, and all other non-outliers, the baseline average will be updated using Eq. 16 and the outlier counter will be reset to 0. In contrast, baseline outliers will be rejected from analysis as possible pile-up and will not be included in baseline averaging.

At initialisation (i.e. in `analinit`) all baseline averages are set to 0.0 and the number of subsequent outliers is reset to 0. Therefore, a small number of pulses is needed to be processed for each detector until the baseline averages come to their nominal value. Note that the adjustment to baseline changes according to this scheme is exponential with an adjustment time-scale τ proportional to `base_avg_fract`.

Finally, the baseline level is checked against a lower and a upper limit, specified by the library parameter variables `minbase` and `maxbase`. If the baseline level violates one of these limits (i.e. baseline < `minbase` or baseline > `maxbase`) the pulse is rejected from analysis. Note that `minbase` and `maxbase` depend on the gain correction.

5.2.7 Determine and check net integral

The pulse net integral is determined using

$$net\_integral = integral − baseline \times shapebins \qquad (18)$$

This net integral, which is taken over the entire range of 96 digitisation bins, is roughly comprised between 0 and 49056 (dependent on gain correction and offset and will be used for further analysis as rough estimate of the event energy value.

`net_integral` is checked against a lower and upper threshold (`minpulse` and `maxpulse`, respectively) in order to reject pulses that lie outside the specified energy range. Current timing measurements showed that roughly 180 μs are needed until this discrimination happens. `minpulse` and `maxpulse` are both specified as 16 Bit unsigned integers in the library parameter block.

5.2.8 Threshold determination

The pulse threshold is determined using

$$threshold = baseline + thresh\_fract \times net\_integral \qquad (19)$$

where `thresh_fract` is the threshold fraction specified in the library parameter block. The pulse threshold is saved in the algorithm variable `threshold`.

PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001 9

5.3 Determine energy dependent parameters

Using net_integral the energy dependent algorithm parameters are extracted from the library parameter block. These are:

- dttpmin: Minimum time-to-peak distance for discrimination
- dttpmax: Maximum time-to-peak distance for discrimination
- maxthresneg: Maximum smaller peak threshold for negative peak spacing
- maxthrespos: Maximum smaller peak threshold for positive peak spacing

In the library parameter block, 10 different values are available for 10 different net_integral reference values for each of the parameters. The energy dependence is resolved by searching the parameter set with the net_integral reference value (or energy reference value) that absolutely comes closest to the measured net_integral value, i.e.

$$min_{i=0,9}\{abs(\text{net\_integral} - \text{libparEnergy}_i)\} \tag{20}$$

5.4 STEP 2 : Pulse filtering

The following tasks are performed in the specified order:

5.4.1 Find pulse starttime, endtime, and duration

The pulse starttime is determined by searching from the ATTP bin down to the first bin with a pulse value that falls below the pulse threshold threshold. The starttime is comprised between 0 and ATTP − 1 and is stored in the algorithm variable starttime.

The pulse endtime is determined by searching from the ATTP bin up to the first bin with a pulse value that falls below the pulse threshold threshold. The endtime is always comprised between ATTP + 1 and 95 (i.e. the last bin) and is stored in the algorithm variable endtime.

The pulse duration is defined as

$$\text{duration} = \text{endtime} - \text{starttime} \tag{21}$$

and is stored in the algorithm variable duration. It is comprised between 2 and 95.

5.4.2 Verify starttime, endtime, and duration

Pulses with starttime within the 'start block' start in the baseline, hence are rejected since they will have a bad baseline. Pulses with endtime within the 'end block' end in the baseline and are also rejected. Pulses with endtime of 95 (i.e. the last bin) are also rejected since they might end well below the pulse limit.

The pulse duration (duration) is checked against a lower and a upper limit, specified in the library parameter block by pulse_dur_min and pulse_dur_max. Typically a pulse is required to last at least for 5 bins and no longer than 60 bins. Pulses outside these limits are rejected.

5.5 STEP 3 : Calculation of inner products and single pulse fitting

The following tasks are performed in the specified order:

5.5.1 Determine number of bins to use

The nominal number of bins to use for the pulse shape analysis is specified in the library control block. Typically, 64 bins are used for the analysis. However, if a pulse appears late in the 96 bins, less than the requested number of bins may be available for the analysis. Thus, the number of bins used for the analysis (n_temp_bins) is limited using

$$\text{n\_temp\_bins} \leq min(64, \text{n\_temp\_bins}, \text{N\_SHAPE\_BINS} - \text{starttime}). \tag{22}$$

On the other hand, the code needs at least 6 bins for correct execution, hence a second constraint

$$6 \leq \text{n\_temp\_bins} \tag{23}$$

is imposed. If this second constraint is violated the pulse is rejected as being too short.

5.5.2 Baseline subtraction

For pulse fitting, the baseline is subtracted from the pulse using an unrolled loop. In order to make the unrolled loop flexible for a variable number of bins, the loop entry point is calculated from **n_temp_bins**. The baseline subtraction algorithm is

$$\text{pulse}_i = \text{pulse}_i - \text{baseline} \tag{24}$$

where $\text{starttime} \leq i < \text{n\_temp\_bins} + \text{starttime}$.

5.5.3 Pulse normalisation

As next step the net pulse integral is calculated using an unrolled loop. The integration algorithm is

$$\text{net\_integral} = \sum_{i=\text{starttime}}^{\text{n\_temp\_bins}+\text{starttime}-1} \text{pulse}_i. \tag{25}$$

Note that **net_integral** now overloads the value calculated earlier (see section 5.2.8). The difference is that in the earlier step, the integral is taken over the entire digitisation length (usually 96 bins) while now the integral is only taken over the number of bins that are used for the fitting (typically 64). In general the difference between both integrals should be small since normally the additional bins should only consist of baseline. However, if some noise or an additional pulse appears in the baseline the integral could be different. If $\text{net\_integral} \leq 0.0$, the pulse is rejected (error code 12). Otherwise, the inverse of the integral is calculated for later pulse normalisation using the subroutine **inverse**.

5.5.4 Inner product and Chi-squared calculation

Inner products between the pulse shape and the templates as well as the best fitting Chi-squared values for single pulse fits are determined in a nested unrolled loop. The result of the inner product calculation are stored in the array **innerproducts**, the best Chi-squared value is stored in **chisqr_best** (note that only χ^2_{mod} is calculated in the code). The inner products are calculated using

$$\text{innerproducts}_j = \frac{1}{\text{net\_integral}} \sum_{i=\text{starttime}}^{\text{n\_temp\_bins}+\text{starttime}-1} \text{pulse}_i \times \text{template}_{i-\text{starttime}, j} \tag{26}$$

where $\text{template}_{i,j}$ is bin i of the normalised template j. The best fitting Chi-squared is then calculated using

$$\chi^2_{\text{mod}} = \text{lib\_x\_lib}_j - 2 \times \text{innerproducts}_j \tag{27}$$

where lib_x_lib_j is the inner product of template j with itself.

The best χ^2_{mod} is evaluated by comparison of the χ^2_{mod} for all templates and stored in **chisqr_best**. The corresponding template index is stored in **best_ttp1**, **ttp1** and **ttp2**, the corresponding scaling factor is set to 1.0 and stored in **alpha_best**. Thus automatically, if no better double pulse fitting can be achieved, the result is a 'double pulse fit' with two identical pulses.

5.6 STEP 4 : Double pulse fitting

The following tasks are performed in the specified order:

5.6.1 Determine extended search indices

Double pulse fitting is done on a limited number of template pairs. For the primary template (here called TTP1) all indices are searched while for the secondary template (here called TTP2) only a limited number of indices are searched defined by

$$max(0, \text{best\_ttp1} - 2) \leq \text{TTP2} \leq min(\text{n\_templates} - 1, \text{best\_ttp1} + 2). \tag{28}$$

Hence double pulse fitting consists of two nested loops where the outer loop goes over TTP2 and the inner, partially unrolled loop, goes over TTP1.

5.6.2 Nested loops

At the beginning of the outer loop, two terms that are invariant to TTP1 are precalculated, i.e.

$$\text{a0} = \text{lib\_x\_lib}_{TTP2} - \text{innerproducts}_{TTP2} \tag{29}$$

and

$$\text{a3} = \text{lib\_x\_lib}_{TTP2} - 2 \times \text{innerproducts}_{TTP2}. \tag{30}$$

The double pulse nominator is then calculated in an unrolled loop using

$$\text{nominator}_{TTP1,TTP2} = \text{a0} + \text{innerproducts}_{TTP1} - \text{lib1\_x\_lib2}_{TTP1,TTP2} \tag{31}$$

with the additional constraint $\text{nominator}_{TTP1} \geq 0.0$. This constraint assures that the amplitude of the primary template is non-negative. The nominator is temporarily stored in the array **nominator**.

The next unrolled loop is used for the calculation of the 'mixing parameter' α using

$$\alpha_{TTP1,TTP2} = \text{nominator}_{TTP1,TTP2} \times \text{denominator}_{TTP1,TTP2}. \tag{32}$$

Note that $\text{denominator}_{TTP1,TTP2}$ stands for the inverse of the denominator d (see Equation 7), i.e.

$$\text{denominator}_{TTP1,TTP2} = \frac{1}{d}. \tag{33}$$

The mixing parameter is temporarily stored in the array **alpha**.

Finally, the best Chi-squared is determined in a normal (non-unrolled loop) using

$$\chi^2_{mod} = \text{a3} - \alpha_{TTP1,TTP2} \times \text{nominator}_{TTP1,TTP2} \tag{34}$$

where a3 is the TTP1-independent part pre-calculated above. In this loop, only $\alpha_{TTP1,TTP2} \leq 1.0$ are considered, assuring that the amplitude of the secondary template is also non-negative. Together with the constraint above it is thus assured that the 'mixing parameter' α is always comprised within [0, 1].

For each TTP2, the best fitting parameters are stored in **chisqr_best**, **alpha_best**, **ttp1**, and **ttp2**, hence at the end of the nested loops these four variables contain the result of the single and double pulse fitting. This means: if the fit using two pulses did not improve upon the fit using a single pulse, both **ttp1** and **ttp2** are identical (signalling a single pulse fit). Otherwise, **ttp1** and **ttp2** are different.

5.7 STEP 5 : Pulse Shape Discrimination

The following tasks are performed in the specified order:

5.7.1 Swap template indices

By definition, **alpha_best** corresponds to the template indexed **ttp1**. Since **alpha_best** is comprised within [0, 1], but both pulses are normalised such that the total area of both pulses is unity, one of the two pulses always has an amplitude of ≤ 0.5. We therefore define that on output the pulse with the smaller amplitude (comprised within [0, 0.5]) is labelled as **ttp1**. Therefore, if **alpha_best** > 0.5 we switch **ttp1** ↔ **ttp2** and set **alpha_best** = 1.0 − **alpha_best**.

5.7.2 Set 15 Bits

The PSD analysis result is compressed into a 15 Bit word using

$$w15 = (alpha\_best \times walpha) \times n\_templates \times n\_templates + ttp2 \times n\_templates + ttp1 + INFO\_MINVAL \quad (35)$$

where

$$walpha = \frac{(INFO\_MAXVAL - INFO\_MINVAL - n\_templates \times n\_templates + 1)}{n\_templates \times n\_templates \times 0.5} \quad (36)$$

and $INFO\_MINVAL = 0x0010$ and $INFO\_MAXVAL = 0x7fff$. The only parameter in this compression scheme is the number of templates used (n_templates). This parameter can be accessed in the PSD housekeeping data, hence onground decompression is assured.

5.7.3 Discrimination

Finally, the pulse shape discrimination verdict is determined using the following logic:

- If $-dttpmin \leq ttp1 - ttp2 \leq dttpmax$ then the pulse is a single pulse
- If $ttp1 - ttp2 < -dttpmin$ and $alpha\_best < maxthresneg$ then the pulse is a single pulse
- If $ttp1 - ttp2 > dttpmax$ and $alpha\_best < maxthrespos$ then the pulse is a single pulse
- Else, the pulse is a multiple

(note that ttp1 corresponds to the smaller pulse while ttp2 corresponds to the larger pulse). Single pulses are flagged by the most significant Bit of the PSD word (Bit 15) set to 0, multiples are flagged by Bit 15 set to 1.

5.8 analinit : Initialisation

This routine performs all necessary actions needed for the initialisation of the scientific software. The actions are (in the specified order):

- Select the floating point rounding mode of the DSP
- Initialise jump table vectors
- Initialise call table vectors
- Evaluate the maximum number of template sets that can fit into the EEPROMs
- Calculate the offset lookup tables used for fast memory access
- Reset baseline averages for all detectors to 0.0

Jump vectors are calculated for the following routines

- Jaddlibrary : Entry point of the addlibrary routine
- Jcorrelate : Entry point of the correlate routine
- Jposterror : Entry point of the posterror routine

The posterror routine is a subroutine that is used locally to post an error message into the housekeeping variables. It is called at various places in addlibrary and correlate.

Call vectors are calculated for the following routines:

- Canal_step0 : Entry point of step 0 of analysis routine
- Canal_step1 : Entry point of step 1 of analysis routine

PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001 13

- Canal_step2 : Entry point of step 2 of analysis routine
- Canal_step3 : Entry point of step 3 of analysis routine
- Canal_step4 : Entry point of step 4 of analysis routine
- Canal_step5 : Entry point of step 5 of analysis routine
- Ceewrite : Entry point of the eewrite routine
- Ccalcoffset : Entry point of the calcoffset routine

The eewrite routine is a subroutine that is used locally to write to the EEPROMs. It is called in addlibrary when a new library vector is added. The calcoffset routine is a subroutine that is used locally to calculate the offset lookup tables used for fast memory access. It is called in analinit.

The maximum number of template sets is evaluated from

$$n\_temp\_sets = frunc\left(\frac{SIZEOFLIBMEM}{SIZEOFSET}\right) \quad (37)$$

where SIZEOFLIBMEM is the size of the available EEPROM memory in Bytes and SIZEOFSET is the size of a template set in Bytes. Both parameters are determined at compile time and usually are SIZEOFLIBMEM = 524288 (= 512 kBytes) and

$$SIZEOFSET = M\_DETS \times SIZEOFLIB = 189696 \quad (38)$$

with

$$SIZEOFLIB = 4 \times (M\_TPLS + 1) \times M\_TEMP\_BINS = 9984 \quad (39)$$

and M_DETS = 19, M_TPLS = 38, and M_TEMP_BINS = 64. Hence, usually 2 template sets may reside in the EEPROMs.

calcoffset calculates the following offset tables (in the given order):

- off_lib_2 = $4 \times$ M_TEMP_BINS $\times i_{tpl}$
- off_lib_3 = $4 \times$ M_TPLS $\times$ M_TEMP_BINS $\times i_{det}$
- off_lEE_3 = SIZEOFLIB $\times i_{det}$
- off_lEE_4 = SIZEOFSET $\times i_{set}$
- off_lxl_2 = $4 \times$ M_TPLS $\times i_{det}$
- off_lxx_2 = $4 \times$ M_TPLS $\times i_{tpl}$
- off_lxx_3 = $4 \times$ M_TPLS $\times$ M_TPLS $\times i_{det}$
- off_lib_p = LIBPARSIZE $\times i_{det}$

where i_{tpl} is the template index, running from 0 to M_TPLS $- 1$ (usually 0-37), and i_{det} is the detector index, running from 0 to M_DETS $- 1$ (usually 0-18). LIBPARSIZE is the size of a library parameter block and amounts to 152 Bytes.

off_lib_2 and off_lib_3 are used to access a library vector in RAM using the template index i_{tpl} and the detector index i_{det}, respectively. The offset of a library template vector with respect to the library base address is calculated using

$$offset = off\_lib\_2[i_{tpl}] + off\_lib\_3[i_{det}]. \quad (40)$$

off_lEE_3 and off_lEE_4 are used to access a library vector in EEPROM using the detector index i_{det} and the set index i_{set}, respectively. The offset of a library template vector with respect to the library base address is calculated using

$$offset = off\_lEE\_3[i_{det}] + off\_lEE\_4[i_{set}]. \quad (41)$$

off_lxl_2 is used to access the inner products among library templates using the detector index i_{det}. The address of a library template vector is calculated using

$$lxlbase = lib\_x\_lib + off\_lxl\_2[i_{det}]. \quad (42)$$

off_lxx_2 and off_lxx_3 are used to access the inner products between library templates in RAM using the template index i_{tpl} and the detector index i_{det}, respectively. The offset of a inner product vector with respect to lib1_x_lib2 is calculated using

$$\texttt{offset} = \texttt{off\_lxx\_2}[i_{tpl}] + \texttt{off\_lxx\_3}[i_{det}]. \tag{43}$$

off_lib_p is used to access the library parameter block using the detector index i_{det}. The address of a library parameter block is calculated using

$$\texttt{parbase} = \texttt{lib\_params} + \texttt{off\_lib\_p}[i_{det}]. \tag{44}$$

5.9　correlate : Pre-calculation

Pre-calculations are done for all 19 detectors and consist of (in the specified order)

- Selection (and verification) of the specified template set using the configuration command parameter

- Extraction (and verification) of the number of bins used for fitting (n_temp_bins)

- Copying of the library parameter block from EEPROM to RAM and verification of the parameters

- Setting of the library validity flag (if all verification were successful)

- Evaluation of walpha

- Copying of templates from EEPROM to RAM, normalising them to unity on the way

- Evaluation of the inner products between and among templates (lib_x_lib and lib1_x_lib2)

- Evaluation of the inverse of the denominator (denominator)

Library templates for the selected library set are copied from EEPROM to RAM. During this copy, the templates are normalised to unity by satisfying the condition

$$1.0 = \sum_{i=0}^{\texttt{n\_temp\_bins}-1} \texttt{template}_{i,TTP1} \tag{45}$$

where n_temp_bins corresponds to the parameter specified in the configuration command (typically 64).

The following arrays are precalculated and stored in RAM:

For TTP1 = 0 to n_templates − 1

$$\texttt{lib\_x\_lib}_{TTP1} = \sum_{i=0}^{\texttt{n\_temp\_bins}-1} \texttt{template}_{i,TTP1} \times \texttt{template}_{i,TTP1} \tag{46}$$

For TTP1 = 0 to n_templates − 1 and TTP2 = 0 to n_templates − 1

$$\texttt{lib1\_x\_lib2}_{TTP1,TTP2} = \sum_{i=0}^{\texttt{n\_temp\_bins}-1} \texttt{template}_{i,TTP1} \times \texttt{template}_{i,TTP2} \tag{47}$$

For TTP1 = 0 to N_TPLS − 1 and TTP2 = 0 to N_TPLS − 1

$$\texttt{denominator}_{TTP1,TTP2} = \frac{1}{\texttt{lib\_x\_lib}_{TTP1} + \texttt{lib\_x\_lib}_{TTP2} - 2 \times \texttt{lib1\_x\_lib2}_{TTP1,TTP2}} \tag{48}$$

5.10 addlibrary : Library upload

Library upload (into EEPROM) is straight forward. On entry, the control fields `libloaddetector`, `libloadcurve`, `libloadset`, and `libloadlength` are checked on validity. If one of the parameters falls out of the allowed range, the routine returns with an error code without writing to EEPROM. If all parameters are valid, the base address of the library vector in EEPROM is calculated using

$$address = libEE + off\_lEE\_4[libloadset] + off\_lEE\_3[libloaddetector] + off\_lib\_2[libloadcurve]. \qquad (49)$$

If `libloadcurve = 255` the library vector is interpreted as algorithm parameter block. In this case the base address is calculated using

$$address = libEE + off\_lEE\_4[libloadset] + off\_lEE\_3[libloaddetector] + off\_lib\_2[M\_TPLS], \qquad (50)$$

i.e. the library parameter block is stored as last template in the EEPROM (this last template is not directly accessible using `libloadcurve = M_TPLS`, hence confusion between a library template and a parameter block is avoided).

A library vector is written to EEPROM using the subroutine `eewrite`. The success of the EEPROM writing is tested by comparing the entire written block in EEPROM to the original data in RAM. If a failure occurred the routine returns with an error code which is transmitted into the housekeeping data.

Before writing a library parameter block, the parameters are checked on validity. If any error occurred, `addlibrary` returns with an error code without writing to EEPROM. If no error occurred, some derived parameters are added (mainly integers which are transformed to floating point values). The entire parameter block is then written to EEPROM using `eewrite`. The success of the EEPROM writing is tested by comparing the entire written block in EEPROM to the original data in RAM. If a failure occurred the routine returns with an error code which is transmitted into the housekeeping data.

6   Memory Management

Four principal memory sections can be discerned in the PSD subassembly:

- ROM : holds the resident copy of the program code
- EEPROM : holds various sets of library templates (uploadable)
- external RAM : holds the active copy of the program code and the pre-calculated arrays
- internal RAM : holds the program variables

While EEPROM is only accessed by the science software, all other memory resources are shared among both eng and science. The following table specifies how these resources are shared:

The 2 HSL buffers can hold 100 events and 5 curves at maximum.

7   Library parameter block

Library parameters are defined by a special library parameter block that is uploaded via library upload using the **curve number 0xff**. The following table specifies the meaning of the parameter block word by word (16-bit words assumed):

8   15 Bit decompression scheme

The following lines show some IDL code that may be used to decompress the PSD information word (stored in PSD16.

```
N_t = 33 ; Number of templates used
INFO_MINVAL = 16
INFO_MAXVAL = 32767
w15 = PSD16 AND 32767
```

16   *PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001*

Table 1: Memory management

memory	start	end	size	package	speed	content
ROM	0x000000	0x007fff	32k	both	slowest	copy of program code
EEPROM A	0x080000	0x0fffff	512k	science	slowest	library templates
ext. RAM	0x980000	0x987fff	32k	both	fast	active copy of program code
ext. RAM	0x988000	0x988fff	4k	eng	fast	event and curve buffer for HSL
ext. RAM	0x989000	0x9efdff	411.5k	science	fast	pre-calculated arrays
ext. RAM	0x9efe00	0x9effff	512	eng	fast	program stack
ext. RAM	0x9f0000	0x9fffff	64k	eng	fast	FIFO buffer for pulses
int. RAM 2	0xffe000	0xffe3ff	1k	eng	fast	program variables
int. RAM 2	0xffe400	0xffe7ff	1k	science	fast	program variables
int. RAM 0	0xfff000	0xfff2ff	768	eng	fast	program variables
int. RAM 0	0xfff300	0xfff7ff	1280	science	fast	program variables
int. RAM 1	0xfff800	0xffffff	2k	eng	fast	program variables

```
if (w15 LT INFO_MINVAL) then begin
print, 'Error-code ',w15
endif else begin wgt_alpha = (INFO_MAXVAL - INFO_MINVAL - N_t*N_t + 1) / (N_t*N_t * 0.5)
w = w15 - INFO_MINVAL
tmp = fix(w/(N_t*N_t))
alpha = tmp/wgt_alpha
ttp2 = fix((w-(tmp*N_t*N_t))/N_t)
ttp1 = fix(w-(tmp*N_t*N_t) - ttp2*N_t)
```

9 Definitions

9.1 Integer variables

- 8-bit signed integers range in value from -128 to 127.

- 16-bit signed integers range in value from -32768 to 32767.

- 16-bit unsigned integers range in value from 0 to 65535.

- 24-bit signed integers range in value from -8388608 to 8388607.

9.2 Times

- ATTP: the bin-index (0-95) of the maximum pulse value

- starttime: the bin-index of the first bin, counted downwards from ATTP, that falls below threshold

- endtime: the bin-index of the first bin, counted upwards from ATTP, that falls below threshold

- duration = endtime − starttime

- pulsettp = ATTP − starttime

9.3 Parameter block and error codes

PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001   17

Table 2: Library parameter block.

index	parameter	meaning
data0a	n_templates	Number of templates in library
data0b	n_start_bins	Number of bins in start block
data0c	n_end_bins	Number of bins in end block
data1a	time_mid	Bin index of mean time
data1b	pulse_dur_min	Minimum pulse duration
data1c	pulse_dur_max	Maximum pulse duration
data2a	base_avg_fract	Average baseline fraction used ×255 (0-255)
data2b	base_outlier	Baseline outlier value
data2c	base_max_outlier	Maximum number of subsequent outliers
data3a	minbase LSB	Minimum baseline (0-511)
data3b	minbase MSB	
data3c	0	empty
data4a	maxbase LSB	Maximum baseline (0-511)
data4b	maxbase MSB	
data4c	0	empty
data5a	minpulse LSB	Minimum pulse area (0-65535)
data5b	minpulse MSB	
data5c	0	empty
data6a	maxpulse LSB	Maximum pulse area (0-65535)
data6b	maxpulse MSB	
data6c	0	empty
data7a	pulse_saturate LSB	Pulse saturation value (0-511)
data7b	pulse_saturate MSB	
data7c	0	empty
data8a	thresh_fraction LSB	Threshold fraction $\times (2^{15} - 1)$ (0-8388607)
data8b	thresh_fraction MID	
data8c	thresh_fraction MSB	
data9a	energy 0 LSB	Pulse area energy bin 0 (0-65535)
data9b	energy 0 MSB	
data9c	0	empty
...	...	...
data18a	energy 9 LSB	Pulse area energy bin 9 (0-65535)
data18b	energy 9 MSB	
data18c	0	empty
data19a	DTTPmin 0	Minimum peak spacing for energy 0 (0-255)
data19b	DTTPmin 1	Minimum peak spacing for energy 1 (0-255)
data19c	DTTPmin 2	Minimum peak spacing for energy 2 (0-255)
data20a	DTTPmin 3	Minimum peak spacing for energy 3 (0-255)
data20b	DTTPmin 4	Minimum peak spacing for energy 4 (0-255)
data20c	DTTPmin 5	Minimum peak spacing for energy 5 (0-255)
data21a	DTTPmin 6	Minimum peak spacing for energy 6 (0-255)
data21b	DTTPmin 7	Minimum peak spacing for energy 7 (0-255)
data21c	DTTPmin 8	Minimum peak spacing for energy 8 (0-255)
data22a	DTTPmin 9	Minimum peak spacing for energy 9 (0-255)
data22b	DTTPmax 0	Maximum peak spacing for energy 0 (0-255)
data22c	DTTPmax 1	Maximum peak spacing for energy 1 (0-255)
data23a	DTTPmax 2	Maximum peak spacing for energy 2 (0-255)
data23b	DTTPmax 3	Maximum peak spacing for energy 3 (0-255)
data23c	DTTPmax 4	Maximum peak spacing for energy 4 (0-255)
data24a	DTTPmax 5	Maximum peak spacing for energy 5 (0-255)
data24b	DTTPmax 6	Maximum peak spacing for energy 6 (0-255)
data24c	DTTPmax 7	Maximum peak spacing for energy 7 (0-255)
data25a	DTTPmax 8	Maximum peak spacing for energy 8 (0-255)
data25b	DTTPmax 9	Maximum peak spacing for energy 9 (0-255)
data25c	0	empty
data26a	maxthres 0 LSB	Maximum smaller peak threshold for negative peak spacing $\times (2^{15} - 1)$ (0-8388607)
data26b	maxthresneg 0 MID	
data26c	maxthresneg 0 MSB	
...	...	...
data35a	maxthresneg 9 LSB	Maximum smaller peak threshold for negative peak spacing $\times (2^{15} - 1)$ (0-8388607)
data35b	maxthresneg 9 MID	
data35c	maxthresneg 9 MSB	
data36a	maxthresneg 0 LSB	Maximum smaller peak threshold for positive peak spacing $\times (2^{15} - 1)$ (0-8388607)
data36b	maxthrespos 0 MID	
data36c	maxthrespos 0 MSB	
...	...	...
data45a	maxthrespos 9 LSB	Maximum smaller peak threshold for positive peak spacing $\times (2^{15} - 1)$ (0-8388607)
data45b	maxthrespos 9 MID	
data45c	maxthrespos 9 MSB	

	I N T E G R A L		SPI-MU-0-1062V3-CNES
cnes		SP I	Issue : 5
CENTRE NATIONAL D'ETUDES SPATIALES	**SPECTROMETER**		Revision : 0
			Date : 28/02/02
			Page No. : ANX9-19

18   *PSD Software Description : SPI-NT-4232-4194-CESR, Edition 2, Revision 1, 9/4/2001*

Table 3: For PSD 15-bit words smaller than INFO_MINVAL the following error codes are defined. The first column gives the code (lowest 15 Bits of the PSD 16 Bit word), the second column describes the error, the third column specifies the state of the most significant Bit of the PSD 16 Bit word (m=multiple=1, s=single=0), and the last column indicates the step of the analysis routine where this error may occur (except of error 12, all error codes correspond to unique break points in the code flow).

code (decimal)	error	type	step
0	No valid library available	m	0
1	Saturated pulse	m	1
2	Pulse area too small	m	1
3	Absolute time-to-peak too early	s	1
4	Absolute time-to-peak too late	s	1
5	Baseline too low	s	1
6	Late pulse starts in baseline	s	2
7	Early pulse ends in baseline	s	2
8	Pulse ends too late	s	2
9	Pulse duration too short	s	2
10	Pulse duration too long	s	2
11	Invalid detector	s	0
12	Pulse area less or equal 0	s	1,3
13	Baseline too high	s	1
14	Baseline outlier	s	1
15	Pulse area too large	m	1

Table 4: The following error codes may be emitted by the PSD subassembly in the HK 0x13 housekeeping structure. The first block (0x01-0x06,0x80) may occur in the eng section of the code. The next three blocks may occur in the science section of the code. The first column gives the error code, the second column desribes the error type, and the third column indicates where the error may occur. For the science errors, all codes correspond to unique break points in the code flow. Errors that arise in the routines addlibrary and eewrite may occur after a library upload. Errors that arise in the routine correlate may occur after sending one of the configuration commands 0x07, 0x08, 0x09.

code (hexadecimal)	error	routine
0x01	No 8 Hz clock	
0x02	Bad 8 Hz clock	
0x03	No HSL	
0x04	Bad library CRC	
0x05	Bad command	
0x06	Bad serial	
0x80	Lost events	
0x30	Curve number too small	addlibrary
0x31	Curve number too large	addlibrary
0x32	Set number too small	addlibrary
0x33	Set number too large	addlibrary
0x34	Detector number too small	addlibrary
0x35	Detector number too large	addlibrary
0x36	Curve length too small	addlibrary
0x37	Curve length too large	addlibrary
0x38	Number of templates too small	addlibrary
0x39	Number of templates too large	addlibrary
0x3a	n_start_bins too small	addlibrary
0x3b	n_start_bins too large	addlibrary
0x3c	n_end_bins too small	addlibrary
0x3d	n_end_bins too large	addlibrary
0x3e	time_mid too small	addlibrary
0x3f	time_mid too large	addlibrary
0x40	Failed to write to EEPROM	eewrite
0x41	Set number too small	correlate
0x42	Set number too large	correlate
0x43	Too few bins used for template	correlate
0x44	Too many bins used for template	correlate
0x45	Too few templates used	correlate
0x46	Too many templates used	correlate
0x47	No valid library template available	correlate
0x48	Too many library templates available	correlate
0x49	Number of bins in shape not 96	correlate
0x4a	Number of bins in start block too small	correlate
0x4b	Number of bins in start block too large	correlate
0x4c	Number of bins in end block too small	correlate
0x4d	Number of bins in end block too large	correlate
0x4e	Mean bin index too small	correlate
0x4f	Mean bin index too large	correlate