Analytical search for optimal flight DIRENA coefficients

Stephan I. Böttcher

2012-03-31, Revision 1040

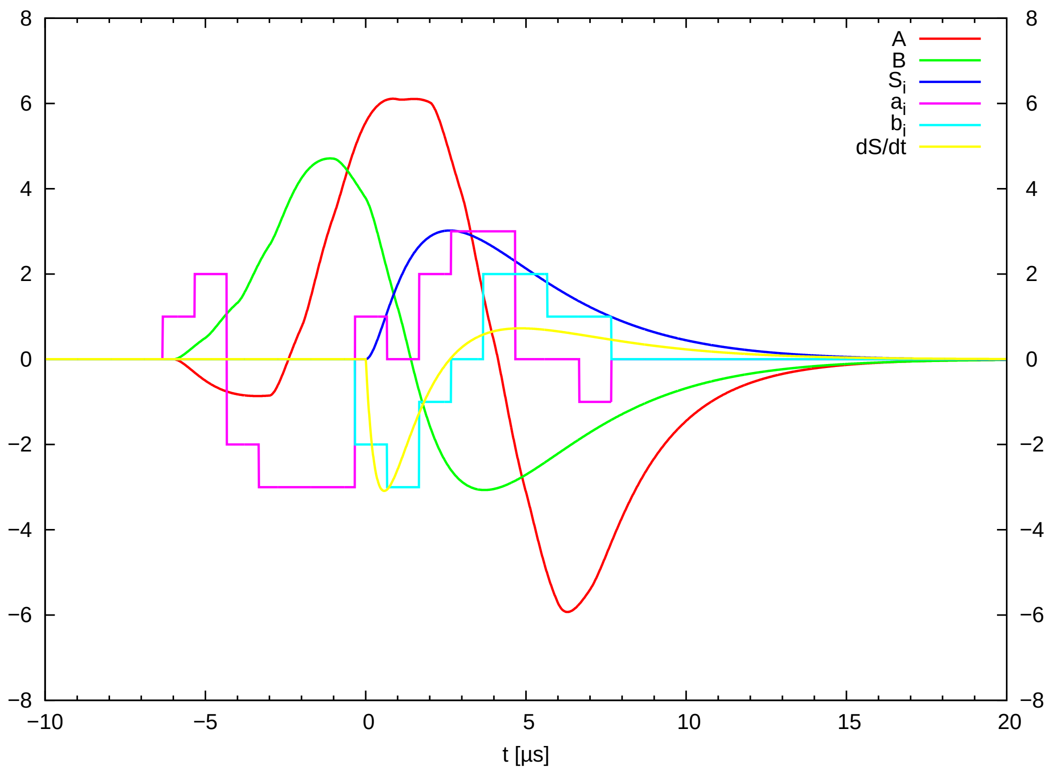

Figure 1: Balanced solution with partial pole-zero correction.
$\tau = 2.2\,\mu$s, $\tau_r = 1000\,\mu$s, $\tau_f = 0.36\,\mu$s,
a_i=$[0, -1, 0, 0, 3, 3, 2, 0, 1, -3, -3, -3, -2, 2, 1, 0]$,
b_i=$[0, 1, 1, 2, 2, 0, -1, -3, -2, 0, 0, 0, 0, 0, 0, 0]$,
$b = 0.360\,\%$, $A_{\max} = 6.110$, $n = 1.27$, $p = -1.6\,\%$.

Contents

List of Figures

1 HET/EPT Flight DIRENA

For the HET/EPT sensor units, a DIRENA type readout is proposed for the silicon detectors and photodiodes. The charge signals will be amplified by discrete preamplifiers, and shaped with one or two shapers. In case of two shapers in parallel, one will have a gain of one for the pulse amplitude, the other one a gain around 16, to increase the dynamic range.

The shaper outputs will be continuously digitized by low power serial ADCs. Further shaping, peak detection and trigger discrimination will be performed digitally by an FPGA.

1.1 Algorithm

The DIRENA computes two linear combinations A and B from a subset of the history of the sampled pulse waveform. s_i shall be the digitization result sampled i conversions in the past, stored in a ring buffer. This computation is performed after every ADC conversion:

$$A = \sum_i a_i s_i \tag{1}$$

$$B = \sum_i b_i s_i \tag{2}$$

with the coefficients a_i and b_i. The sums of the coefficients are normalized to zero:

$$\sum_i a_i = 0 \tag{3}$$

$$\sum_i b_i = 0 \tag{4}$$

so that the linear combinations are zero in the absents of a pulse, and proportional to the pulse height when a pulse is present at a given phase.

The coefficients shall be chosen such that A becomes large for a pulse close to the nominal phase, while B is near zero for the same phase. A shall not vary much with the phase of the pulse, while B depends strongly on the phase. A will be used to estimate the pulse height, the ratio B/A allows to estimate the phase of the pulse.

The sequence of A-s obtained will be tested for a maximum in time, and compared to a discriminator thresholds to trigger further processing.

1.2 Flight vs Altera DIRENA

The flight DIRENA for Solar Orbiter HET/EPT shall be implemented in an RTAX2000 FPGA, which does not provide dedicated multiplier cells, like the Altera FPGA does for our non-flight DIRENA. The DIRENA algorithm was modified so it can be implemented efficiently without multipliers. The differences are:

- The analysis coefficients are integers in the range $[-3 \ldots 3]$, instead of [-4095. . . 4095].
- All channels use the same set of coefficients.
- The analysis is applied to 15 consecutive samples. The Altera implementation can use a set of 16 samples out of 64 consecutive samples.
- The pulse shape is sampled at 1 MSPS instead of 3 MSPS, because the flight ADC cannot run faster than that.

The pulses shall be shaped with a shaping time constant of $\tau = 2.2\,\mu$s.

1.3 Hardwired configuration

The flight FPGAs may have the coefficients hardwired at compile time. The advantages are:

- reduced complexity of the configuration procedure,
- no SEU crossection,
- less logic, and
- less power consumption.

1.4 Optimization criteria

1.4.1 Banana error

The estimated pulse height A depends on the ADC clock phase at the time of the pulse arrival. The ratio of A to the actual pulse height plotted versus the phase usually looks like a banana with a maximum somewhere in the center and two ends pointing down by the same amount, as illustrated in Fig. 2.

Since the phase can be computed from the ratio B/A, the banana error can be corrected. This is typically done offline in a PC for precision analysis of PHA data, but cannot efficiently be done inside the FPGA for flight.

The coefficients a_i shall be chosen such that the banana error is within acceptable limits. The banana parameter b is computed from the maximum

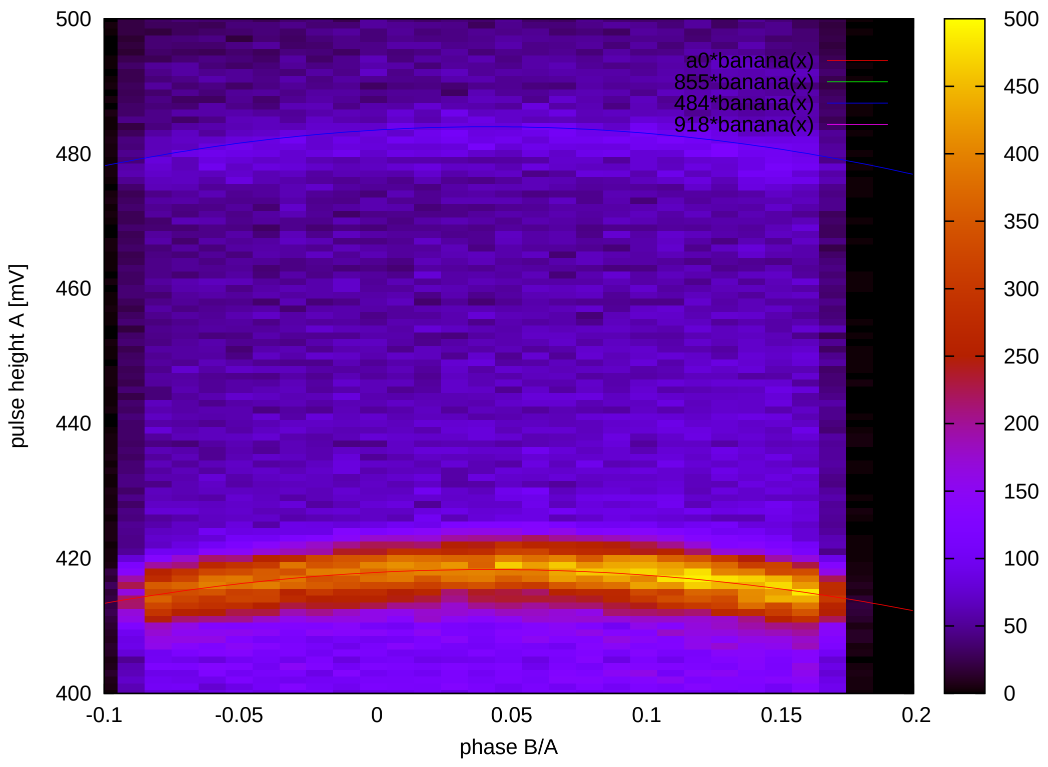

Figure 2: Banana of a non-flight IRENA for ^{207}Bi conversion electron lines. The shaping time is $\tau = 1\,\mu$s, sampled at 3 MSPS.

A for any phase A_{max}, and the largest A_{min} for which all $A > A_{\mathrm{min}}$ in a range around the maximum for a phase range of at least one ADC clock period

$$b = \frac{A_{\mathrm{max}} - A_{\mathrm{min}}}{A_{\mathrm{max}}}. \tag{5}$$

1.4.2 Noise

The estimated pulse height A shall be computed such that the electronic noise contribution is minimized. Assuming independent noise of each sample, the noise parameter n is computed as

$$n = \frac{\sqrt{\sum a_i^2}}{A_{\mathrm{max}}}. \tag{6}$$

1.4.3 Pileup

For high particle rates it is important that A returns to zero fast when the pulse ages out of the nominal phase in the ring buffer, so that the pulse height estimate for a new pulse is not offset by the tail of the previous pulse.

The parameter p used here to estimate this property is the fraction of A for a pulse analyzed $15\,\mu$s later than when it hit the $A_{\min}$ range

$$p = \frac{A(15\,\mu\text{s})}{A_{\max}}. \tag{7}$$

p is normally negative, because the pulse samples will hit the negative baseline coefficients a few clocks after the maximum, and then return to zero. When the shaper output is bipolar, or the coefficients are improperly chosen, the A will overshoot the zero line eventually. This overshoot shall be smaller than any reasonable trigger threshold even for maximum size pulses.

A pole-zero compensation in the shaper may be required as well to avoid the overshoot.

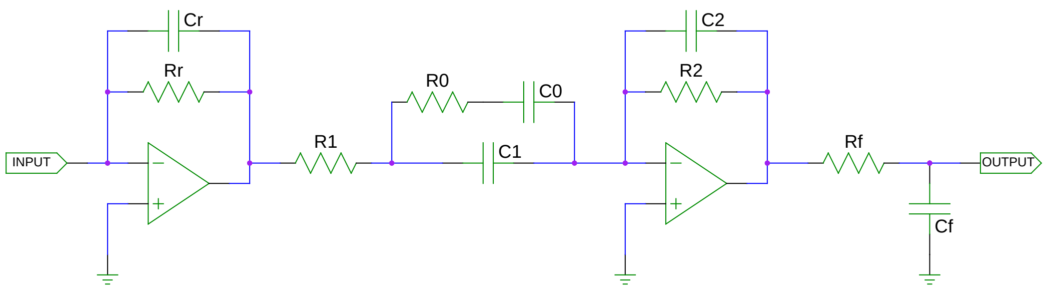

Figure 3: Schematic diagram of the frontend. For the initial analysis the pole-zero compensation circuit $C_0 - R_0$ was not present.

2 Shaper pulse shape

The shaper output pulse shape will be computed analytically from the inverse Laplace transform of the response functions of the preamplifier, the shaper, and the ADC input filter.

2.1 Preamplifier

The preamplifier deposits the input charge pulse on its feedback capacitor, which is discharged by the feedback resistor with the time constant $\tau_r = 100\,\mu$s. The pulse response is

$$h_r(t) = \frac{1}{C_r}\Theta(t)\,\exp(-t/\tau_r). \tag{8}$$

The transfer function is the Laplace transform of the pulse response:

$$H_r(s) = \mathcal{L}\left\{h(t)\right\} = \frac{1}{C_r}\frac{\tau_r}{1+s\tau_r}. \tag{9}$$

2.2 Shaper

The shaper is a current feedback operational amplifier, operating as an inverting amplifier with gain

$$V = \frac{Z_2}{Z_1} \tag{10}$$

Z_1 is the complex impedance of the input branch, a resistor R_1 and a capacitor C_1 in series, with time constant $\tau = R_1C_1 = 2.2\,\mu$s

$$Z_1 = R_1 + \frac{1}{j\omega C_1} = R_1\frac{1+j\omega\tau}{j\omega\tau}. \tag{11}$$

Z_2 is the impedance of the feedback branch, a resistor R_2 and a capacitor C_2 in parallel, with the same time constant $\tau = R_2C_2 = 2.2\,\mu$s

$$Z_2 = \frac{\frac{R_2}{j\omega C_2}}{R_2 + \frac{1}{j\omega C_2}} = \frac{R_2}{1+j\omega\tau} \tag{12}$$

The transfer function of the shaper is V, with $j\omega = s$

$$H_s(s) = \frac{Z_2}{Z_1} = \frac{as\tau}{(1+s\tau)^2} \tag{13}$$

with the linear amplification factor $a = R_2/R_1 = C_1/C_2$.

2.3 ADC input filter

Between shaper output and ADC input there is a simple *R*-*C* filter. The resistor R_f is located close to the opamp, to limit its capacitive load to a minimum. The capacitor C_f is located close to the ADC input, to capture any charge that the ADC input will emit with the start of each conversion when the track and hold captures the input voltage. The filter has a time constant $\tau_f = R_fC_f = 0.33\,\mu$s. The transfer function is that of a voltage divider built from $Z_3 = R_f$ and $Z_4 = 1/(j\omega C_f)$:

$$H_f(s) = \frac{Z_4}{Z_3+Z_4} = \frac{\frac{1}{j\omega C_f}}{R_f + \frac{1}{j\omega C_f}} = \frac{1}{1+s\tau_f} \tag{14}$$

2.4 Frontend transfer function

The transfer function of the frontend electronics is the product of its parts

$$\begin{aligned} H(s) &= H_r(s)H_s(s)H_f(s) && (15) \\ &= \frac{1}{C_r}\,\frac{\tau_r}{1+s\tau_r}\,\frac{as\tau}{(1+s\tau)^2}\,\frac{1}{1+s\tau_f} && (16) \end{aligned}$$

2.5 Inverse Laplace transform

The silicon detectors are very fast compared to the electronics, the charge pulses are considered δ-functions in time:

$$q(t) = q\delta(t) \tag{17}$$

The output pulse shape is thus obtained directly by the inverse Laplace transform of the transfer function.

$$u(t) = \mathcal{L}^{-1}\left\{H(s)\right\}. \tag{18}$$

The computer algebra program maxima was used to compute the inverse Laplace transform.

```
maxima
display2d:false$
H: s/(1+s*T)**2/(1+s*TR)/(1+s*TP);
h: ilt(H,s,t);
 -TR*%e^-(t/TR)/(TR^3+(-TP-2*T)*TR^2+(2*T*TP+T^2)*TR-T^2*TP)
      -%e^-(t/T)*(T*TP*TR-T^3)/(T*((TP^2-2*T*TP+T^2)*TR^2
                                +(-2*T*TP^2+4*T^2*TP-2*T^3)*TR+T^2*TP^2
                                -2*T^3*TP+T^4))
      +TP*%e^-(t/TP)/((TP^2-2*T*TP+T^2)*TR-TP^3+2*T*TP^2-T^2*TP)
      -t*%e^-(t/T)/(T*((TP-T)*TR-T*TP+T^2))
```

This output could be copied and pasted into gnuplot and python scripts after minor syntax fixes. Some transformations yield:

$$\begin{aligned} h_1(t) &= -\exp(-t/\tau_r)\,\frac{\tau_r}{(\tau_r-\tau_f)(\tau_r-\tau)^2} && (19) \\ h_2(t) &= -\exp(-t/\tau)\,\frac{\tau_f\tau_r-\tau^2}{(\tau_f-\tau)^2(\tau_r-\tau)^2} && (20) \\ h_3(t) &= \exp(-t/\tau_f)\frac{\tau_f}{(\tau_f-\tau)^2(\tau_r-\tau_f)} && (21) \\ h_4(t) &= -t\exp(-t/\tau)\frac{1}{\tau\tau_f\tau_r-\tau^2(\tau_r+\tau_f)+\tau^3} && (22) \\ h(t) &= \frac{a\tau\tau_r}{C_r}\,(h_1(t)+h_2(t)+h_3(t)+h_4(t)) && (23) \end{aligned}$$

2.6 Pole-zero compensation

By adding a resistor R_0 parallel to C_1, the preamplifier time constant can be compensated in the shaper. For now we consider the capacitance of C_0 to be infinite, i.e., shorted. The impedance Z_1 becomes:

$$Z_1 = R_1 + \frac{R_0 \frac{1}{j\omega C_1}}{R_0 + \frac{1}{j\omega C_1}} = R_1 \frac{\tau_r}{\tau} \frac{1 + j\omega\tau}{1 + j\omega\tau_r}, \tag{24}$$

with $\tau_r = C_1 R_0$ and $\tau = C_1 R_0 R_1 / (R_0 + R_1)$. The transfer function of the shaper with R_0 is

$$H_s(s) = \frac{Z_2}{Z_1} = \frac{a\tau}{\tau_r} \frac{1 + s\tau_r}{(1 + s\tau)^2}. \tag{25}$$

In the overall transfer function, the pole of the preamplifier cancels with the new zero in the shaper

$$H(s) = \frac{1}{C_r} \frac{a\tau}{(1 + s\tau)^2} \frac{1}{1 + s\tau_f}. \tag{26}$$

The pulse response is now:

$$h(t) = \frac{a}{C_r} \left(\left(e^{-\frac{t}{\tau_f}} - e^{-\frac{t}{\tau}} \right) \frac{\tau\tau_f}{(\tau - \tau_f)^2} + t e^{-\frac{t}{\tau}} \frac{1}{\tau - \tau_f} \right) \tag{27}$$

2.7 Pole-zero compensation, AC-coupled

The resistor R_0 introduces a DC-gain to the circuit. The preamplifier outputs have a DC level at about $V_{\mathrm{GS}} = -250\,\mathrm{mV}$, that is the gate-source voltage of the jFET at the chosen bias current. R_0 gives a DC-gain of one to a high-gain shaper. The resulting DC offset will cut into the dynamic range of the channel. For DC-coupled detectors, the detector bias current will shift the DC level further. That could be considered a nice housekeeping diagnostic. But we do not plan to use DC-coupled detectors, because of the associated loss of dynamic range at the preamplifier output.

To avoid the DC-gain, the capacitor C_0 shall AC-couple the pole-zero compensation. The capacitance of C_0 shall be as large as possible, for flight, in size 0805, the largest available capacitance is $C_0 = 220\,\mathrm{nF}$, compared to a high-gain $C_1 = 10\,\mathrm{nF}$ this is not negligible.

So let us look at the transfer function with C_0 included. The input impedance of the shaper becomes:

$$Z_1 \;=\; R_1 + \frac{\frac{1}{j\omega C_1}\left(R_0 + \frac{1}{j\omega C_0}\right)}{\frac{1}{j\omega C_1} + R_0 + \frac{1}{j\omega C_0}} \tag{28}$$

$$
\begin{aligned}
&= R_1 \frac{1 + s(\tau_1 + \tau_{00}) + s^2 \tau_1 \tau_r}{(1 + s\tau_r) s\tau_1} && (29) \\
&= \frac{R_1}{s\tau_1} \frac{(1 + s\tau)(1 + s\tau_0)}{1 + s\tau_r} && (30)
\end{aligned}
$$

with

$$
\begin{aligned}
\tau_1 &= (C_0 + C_1) R_1 && (31) \\
\tau_{00} &= C_0 R_0 && (32) \\
\tau_r &= C_1 R_0 \, C_0 / (C_0 + C_1) && (33) \\
\tau \tau_0 &= \tau_1 \tau_r = C_0 C_1 R_0 R_1 && (34) \\
\tau + \tau_0 &= \tau_1 + \tau_{00} = R_1 C_0 + R_1 C_1 + R_0 C_0 . && (35)
\end{aligned}
$$

This is a system of equations that needs to be solved for R_0, R_1, and τ_0 to implement the desired pole $(1 + s\tau)$ and zero $(1 + s\tau + r)$ in the shaper transfer function

$$
\begin{aligned}
R_0 &= \frac{\tau_r}{C_1} \left(1 + \frac{C_1}{C_0} \right)^{-1} , && (36) \\
R_1 &= \frac{\tau}{C_1} \left(1 + \frac{\tau}{\tau_r} + \mathcal{O}\left\{ \frac{\tau}{\tau_0} \right\} \right) , && (37) \\
\tau_0 &= C_0 R_0 \left(1 + \frac{R_1}{R_0} + \mathcal{O}\left\{ \frac{\tau^2}{\tau_r \tau_0} \right\} \right) . && (38)
\end{aligned}
$$

The zero will cancel the pole of the preamplifier, which is replaced by the a new pole $(1 + s\tau_0)$ with a much larger time constant. The structure of the transfer function is the same as the case without pole-zero compensation, so we can use the function from section 2.5, just with a larger value for t_r.

With $C_1 = 10\,\mathrm{nF}$ and $C_0 = 220\,\mathrm{nF}$ the new parasitic pole with have the time constant $\tau_r = 2.2\,\mathrm{ms}$.

2.8 IRENA $1\,\mu$s pulse shape

Fig. 4 demonstrates how well the inverse Laplace transform fits to the data sampled with the non-flight IRENA. The ADC samples were normalized to the reconstructed phase and banana-corrected amplitude. The time constants were fitted to the data.

3 Banana optimized coefficients

A python script was developed to search for coefficients a_i that minimize the banana error.

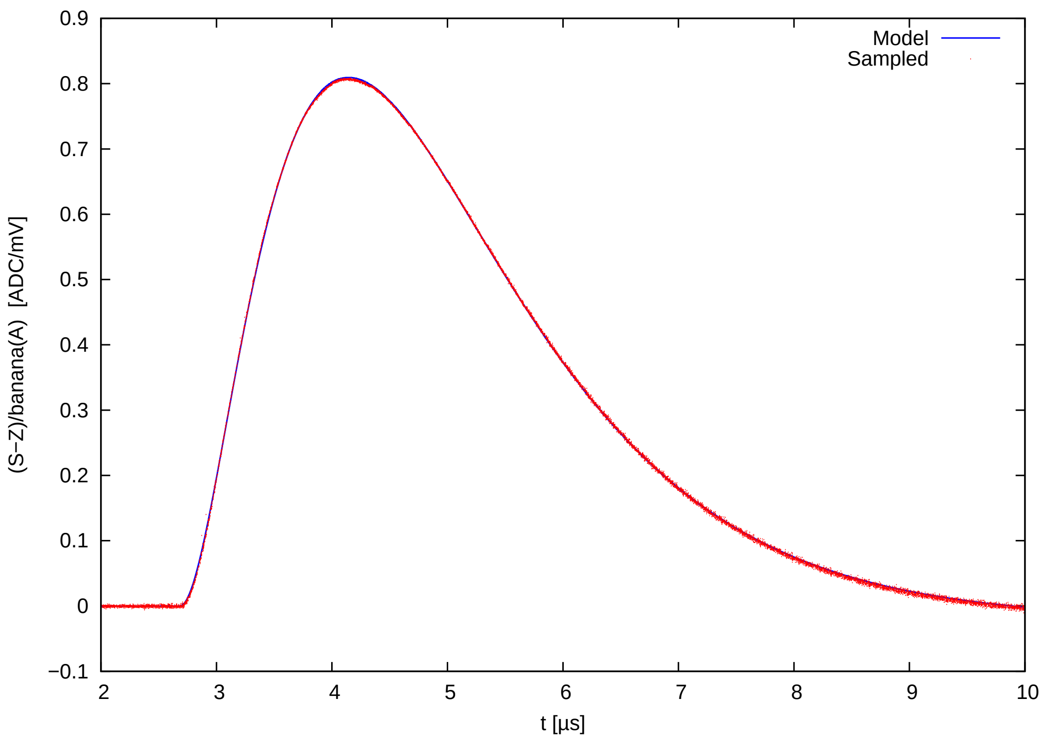

Figure 4: Non-flight pulse shape, sampled and modeled.
$\tau = 1.02\,\mu\text{s}$, $\tau_r = 110.8\,\mu\text{s}$, $\tau_f = 0.363\,\mu\text{s}$.

3.1 Constraints on a_i

The coefficients for the latest sample a_0, b_0 must be zero, because the FPGA implementation cannot at this point reliably use that ADC sample. It is doubtful if that can be fixed. So there are only 15 samples available, $a_1 \dots a_{15}$. (The python script uses a reversed sample index.)

The value space of eight coefficients was searched. The remaining seven coefficients were used to normalize the sum to zero. The ranges of values considered were:

- $a_0 = 0$.
- a_1 and a_2 were allowed to take values $[-1, 0, 1]$. These sample the tail of the pulse.
- $a_3 \dots a_5$ were searched in the range $[0 \dots 3]$, to sample the peak of the pulse and obtain good sensitivity for the amplitude.
- a_6 was forced to be non-zero, $[1 \dots 3]$, to anchor the pulse reconstruction phase to a fixed ADC clock period.

- a_7 is either 0 or 1. This coefficient samples the steepest part of the pulse. Larger values would cause larger banana errors. After some experiments, the values 2 and 3 were omitted for efficiency from further searches.

- $a_8 = 1$ is fixed. Experiments have shown that all good solutions need this value.

The next two or three coefficients were were given the value -3, and one further negative value, until the overall sum was zero. Those coefficients sample the baseline of the shaper output before the pulse.

The coefficients b_i were derived manually by trial and error until a suitably linear dependency on the phase in the right range resulted. The values are listed in the caption of Fig. 1.

3.2 Results

For each tested set of coefficients the banana error b was calculated. All solutions with $b < 1\,\%$ were saved, in a list, sorted, and manually reviewed.

For $\tau = 2.2\,\mu$s, $\tau_r = 1000\,\mu$s, and $\tau_f = 0.36\,\mu$s, the best 50 and some notable results are:

```
0.00281 0.700 6.960 -0.084 1.06 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 3, 0, 1, 1, 0]
0.00287 0.670 6.818 -0.087 1.04 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 2, 1, 1, 1, 0]
0.00305 0.660 6.703 -0.089 1.12 [0, 0, 0, -2, -3, -3, -3, 1, 1, 2, 0, 3, 3, 1, 0, 0]
0.00317 0.650 6.626 -0.075 1.07 [0, 0, 0, -1, -3, -3, -3, 1, 1, 1, 2, 3, 2, 1, -1, 0]
0.00330 0.740 6.407 -0.059 1.13 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 2, 1, 3, 0, -1, 0]
0.00336 0.680 6.187 -0.063 1.12 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 3, 1, 0, 0, 0]
0.00336 0.670 6.648 -0.074 1.11 [0, 0, 0, -1, -3, -3, -3, 1, 1, 1, 2, 3, 3, -1, 0, 0]
0.00348 0.660 7.099 -0.083 1.04 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 3, 1, 0, 1, 0]
0.00348 0.710 6.682 -0.057 1.12 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 2, 2, 3, -1, -1, 0]
0.00348 0.630 6.679 -0.090 1.06 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 1, 2, 1, 1, 0]
0.00354 0.640 6.959 -0.086 1.04 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 2, 2, 0, 1, 0]
0.00360 0.670 6.110 -0.052 1.11 [0, 0, 0, 0, -2, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
0.00360 0.730 6.599 -0.073 1.09 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 0, 2, 0, 1, 0]
0.00366 0.630 6.767 -0.073 1.09 [0, 0, 0, -1, -3, -3, -3, 1, 1, 1, 2, 3, 3, 0, -1, 0]
0.00385 0.610 6.487 -0.077 1.09 [0, 0, 0, -1, -3, -3, -3, 1, 1, 1, 2, 2, 3, 1, -1, 0]
0.00385 0.660 6.327 -0.062 1.14 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 3, 2, -1, 0, 0]
0.00385 0.710 6.874 -0.070 1.07 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 1, 2, -1, 1, 0]
0.00391 0.750 6.551 -0.057 1.08 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 2, 2, 2, 0, -1, 0]
0.00397 0.700 6.734 -0.072 1.13 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 0, 3, -1, 1, 0]
0.00397 0.730 6.825 -0.055 1.10 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 2, 3, 2, -1, -1, 0]
0.00403 0.610 6.820 -0.089 1.08 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 1, 3, 0, 1, 0]
0.00403 0.650 6.047 -0.065 1.12 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 2, 2, 0, 0, 0]
0.00403 0.740 7.261 -0.093 1.05 [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 1, 3, 1, 1, 0]
0.00409 0.780 6.468 -0.073 1.09 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 0, 1, 1, 1, 0]
0.00415 0.650 6.251 -0.051 1.11 [0, 0, 0, 0, -2, -3, -3, 1, 0, 2, 3, 3, 1, -1, -1, 0]
0.00415 0.750 6.742 -0.071 1.05 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 1, 1, 0, 1, 0]
0.00415 0.710 6.995 -0.069 1.03 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 2, 0, 1, 0, 0]
0.00421 0.680 6.716 -0.073 1.07 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 0, 2, 1, 0, 0]
0.00421 0.720 7.536 -0.090 1.03 [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 2, 3, 0, 1, 0]
0.00421 0.590 6.541 -0.094 1.12 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 0, 3, 1, 1, 0]
0.00427 0.710 6.055 -0.064 1.14 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 3, 0, 1, 0, 0]
0.00427 0.640 5.768 -0.069 1.15 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 1, 2, 1, 0, 0]
0.00427 0.660 5.833 -0.055 1.11 [0, 0, 0, 0, -2, -3, -3, 1, 0, 2, 3, 2, 0, 1, -1, 0]
0.00427 0.690 6.855 -0.071 1.03 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 1, 1, 1, 0, 0]
0.00427 0.720 7.017 -0.068 1.05 [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 2, 1, -1, 1, 0]
0.00427 0.670 5.911 -0.067 1.12 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 2, 1, 1, 0, 0]
0.00433 0.760 7.214 -0.079 1.06 [0, 0, 0, -2, -3, -3, -3, 1, 1, 2, 1, 3, 3, 1, -1, 0]
0.00433 0.640 6.309 -0.063 1.12 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 3, 1, 1, -1, 0]
0.00433 0.670 7.651 -0.090 1.01 [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 2, 3, 1, 0, 0]
0.00433 0.640 5.971 -0.054 1.09 [0, 0, 0, 0, -2, -3, -3, 1, 0, 2, 3, 2, 1, 0, -1, 0]
0.00439 0.770 6.695 -0.056 1.08 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 2, 3, 1, 0, -1, 0]
```

```
0.00439 0.690 7.868 -0.100 1.09 [0, 0, 0, -4, -3, -3, -3, 1, 1, 3, 0, 3, 3, 1, 1, 0]
0.00446 0.680 7.791 -0.088 0.99 [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 3, 2, 1, 0, 0]
0.00446 0.700 7.812 -0.087 1.04 [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 3, 3, -1, 1, 0]
0.00452 0.630 5.691 -0.058 1.11 [0, 0, 0, 0, -2, -3, -3, 1, 0, 2, 3, 1, 1, 1, -1, 0]
0.00452 0.620 7.239 -0.083 1.05 [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 3, 2, -1, 1, 0]
0.00458 0.770 7.159 -0.065 1.08 [0, 0, 0, -1, -3, -3, -3, 1, 1, 1, 3, 3, 3, -1, -1, 0]
0.00458 0.630 6.188 -0.064 1.17 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 1, 2, 3, -1, 0, 0]
0.00458 0.750 7.405 -0.091 1.01 [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 2, 2, 1, 1, 0]
0.00458 0.740 6.134 -0.062 1.18 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 2, 0, 3, 1, -1, 0]
0.00458 0.770 6.278 -0.060 1.10 [0, 0, 0, 0, -3, -3, -3, 1, 0, 3, 2, 1, 2, 1, -1, 0]

0.00970 0.830 9.159 -0.080 0.96 [0, 0, 0, -4, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]

0.00995 0.730 5.434 -0.046 1.16 [0, 0, 0, 0, -1, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]
```

The columns are: the banana error b, the phase where $A_{\min}$ is first reached, $A_{\max}$, The pileup parameter p, and the noise parameter n, followed by the list of coefficients $[a_{15} \dots a_0]$.

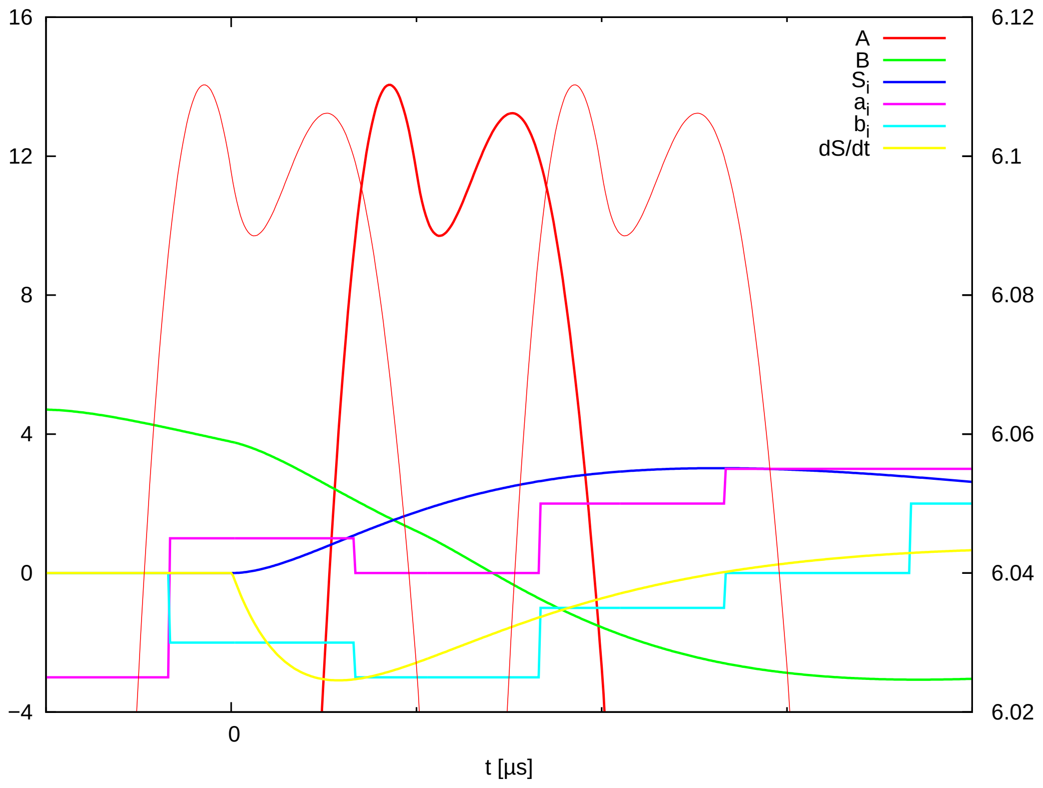

Figure 5: Double peaked pulse height vs phase. Same solution as in Fig. 1, zoomed into the A-graph (right y-axis). The graph is also shown shifted by ± 1 ADC clock period on the x-axis. The crossings mark the range of phase used for pulse height analysis.

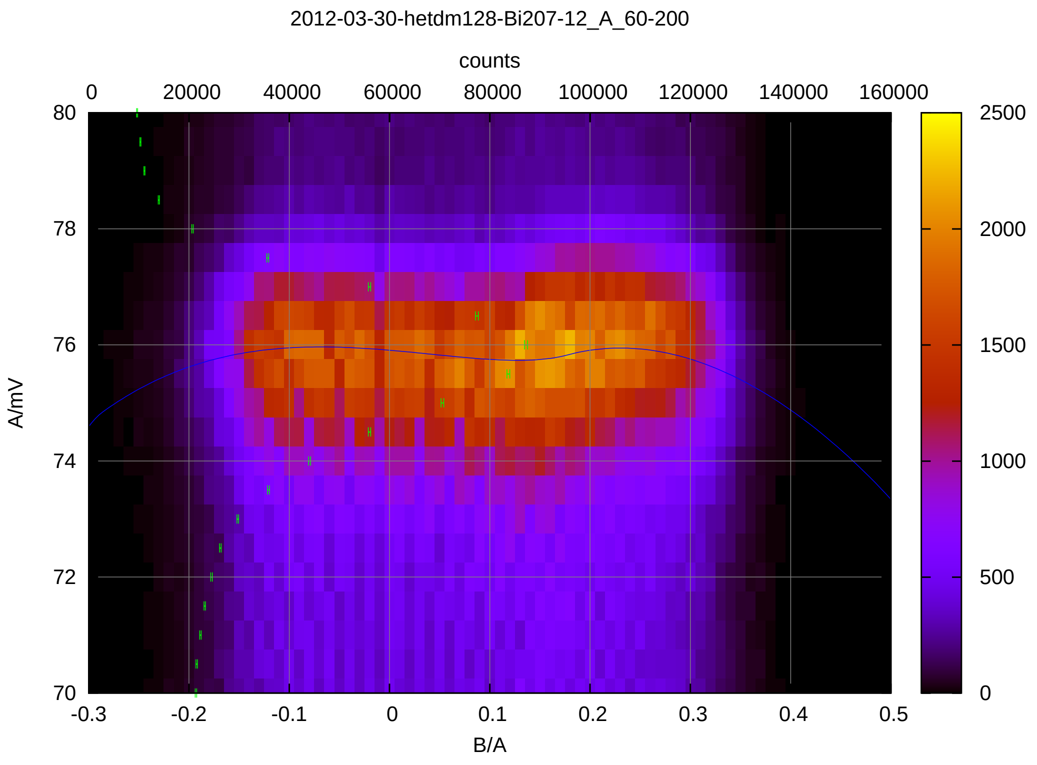

Figure 6: Banana of the flight IRENA test with the HET demonstrator, for a ^{207}Bi conversion electron line. The shaping time is $\tau = 2.2\,\mu$s, sampled at 1 MSPS, no pole-zero compensation. Also shown is the analytic banana model.

3.2.1 How to get a good Banana

To get a low banana error b, the pulse height estimate A must assume two maxima that are less than one ADC clock period apart in phase. Negative coefficients in the tail tend to help, as well are non-monotonic patterns of coefficient on the peak. The negative baseline coefficients have no influence on the banana error at all.

3.2.2 How to get low noise

For low noise, the coefficients along on the pulse peak must be maximized, to yield a high A_{max}. The result at $b = 0.970\,\%$ gives the best noise parameter $n = 0.96$ (for $b < 1\,\%$). Note the large $A_{\text{max}} = 9.159$.

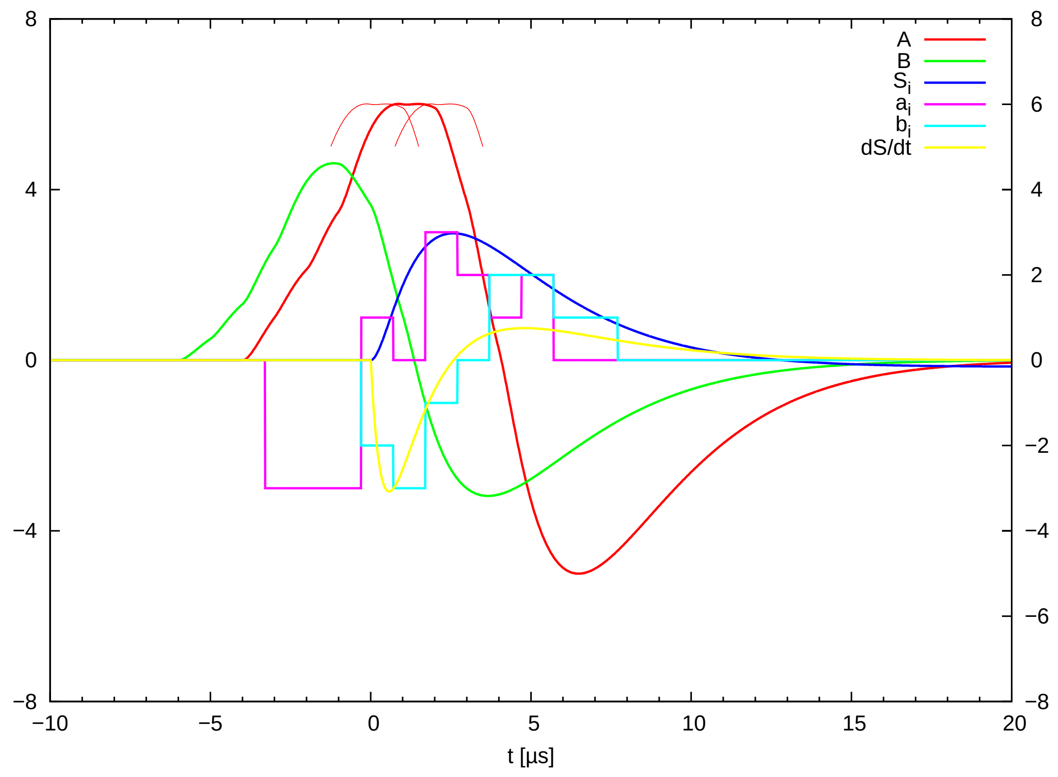

Figure 7: Solution used for the HET test.
$\tau = 2.2\,\mu\text{s}$, $\tau_r = 100\,\mu\text{s}$, $\tau_f = 0.36\,\mu\text{s}$,
a_i=[0, 0, 0, 0, −3, −3, −3, 1, 0, 3, 2, 1, 2, 0, 0, 0],
b_i=[0, 0, 0, 0, 0, 0, 0, −2, −3, −1, 0, 2, 2, 1, 1, 0],
$b = 0.317\,\%$, $A_{\max} = 6.009$, $n = 1.13$, $p = -6.2\,\%$.

3.2.3 How to optimize for pileup

Fast recovery for low pileup requires narrow coefficient patterns. This conflicts with the desire for low noise and low banana error.

Another way to get fast recovery is to add positive coefficients on the baseline preceding the negatively weighed baseline samples. This pileup tuning must be carefully applied to avoid overshoots. Overshoots may cause extra triggers after large pulses, which must be avoided. The pileup tuning adds to the coefficients without yielding larger $A_{\max}$, so the pileup improvements come at the cost of additional electronic noise.

For example, tuning the result with the best native pileup from the list, with $b = 0.995\,\%$, yields:

```
0.00995 0.730 5.434 -0.046 1.16 [0, 0, 0, 0, -1, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]
0.00995 0.730 5.434 -0.034 1.33 [0, 0, 0, 2, -3, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]
0.00995 0.730 5.434 -0.026 1.30 [0, 0, 1, 1, -3, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]
```

```
0.00995 0.730 5.434 -0.018 1.33 [0, 0, 2, 0, -3, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]
0.00995 0.730 5.434 -0.010 1.40 [0, 0, 3, -1, -3, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]
0.00995 0.730 5.434 0.009 1.45 [0, 1, 3, -2, -3, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]
```

The last result causes an overshoot of $1\,\%A_{\text{max}}$, and is not suitable.

Pileup tuning of the result with best noise, at $b = 0.97\,\%$, yields:

```
0.00970 0.830 9.159 -0.084 0.93 [0, 0, -1, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
0.00970 0.830 9.159 -0.078 0.95 [0, 1, -2, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
0.00970 0.830 9.159 -0.072 1.00 [0, 2, -3, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
0.00970 0.830 9.159 -0.064 0.99 [1, 1, -3, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
0.00970 0.830 9.159 -0.055 1.00 [2, 0, -3, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
0.00970 0.830 9.159 -0.047 1.04 [3, -1, -3, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
```

This would be candidate when the emphasis is to be both fast and low noise, but with a rather large banana error. This solution employs the full range of coefficients, and could probably even use more, if they were available.

The solution presented in Fig. 1 is derived from a rather low-b and low-p result:

```
0.00360 0.670 6.110 -0.052 1.11 [0, 0, 0, 0, -2, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
0.00360 0.670 6.110 -0.047 1.18 [0, 0, 0, 1, -3, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
0.00360 0.670 6.110 -0.040 1.18 [0, 0, 1, 0, -3, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
0.00360 0.670 6.110 -0.033 1.22 [0, 0, 2, -1, -3, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
0.00360 0.670 6.110 -0.016 1.27 [0, 1, 2, -2, -3, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
0.00360 0.670 6.110 -0.008 1.37 [0, 1, 3, -3, -3, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
```

The last line produces an overshoot of $0.01\,\%$. An overshot that low may be acceptable, but the fact that it is present suggests to take a step back.

4 HET demonstrator test

The ADC test board was connected to the HET demonstrator sensor head, configured with banana-optimized coefficients, and tested with a ^{207}Bi source. The thick window of the sensor head degrades the intrinsic resolution of the obtained electron lines, but the shape of the banana can be observed in Fig. 6.

The coefficients and model parameters are given in the caption of Fig. 7.

The measured banana error appears to be larger than modeled. Reasons may be that the real preamplifier time constant is $\tau_r = 330\,\mu\text{s}$, because the feedback resistor was changed from $C_r = 1\,\text{pF}$ to $C_r = 3.3\,\text{pF}$ without adapting the feedback resistor.

5 Hamamatsu diode test, with pole-zero compensation

The ADC test board was connected to a preamplifier test board with five preamplifiers, two of those are populated with flight UHF transistors. Pole-zero corrections were added to three shapers on the ADC test board, all with

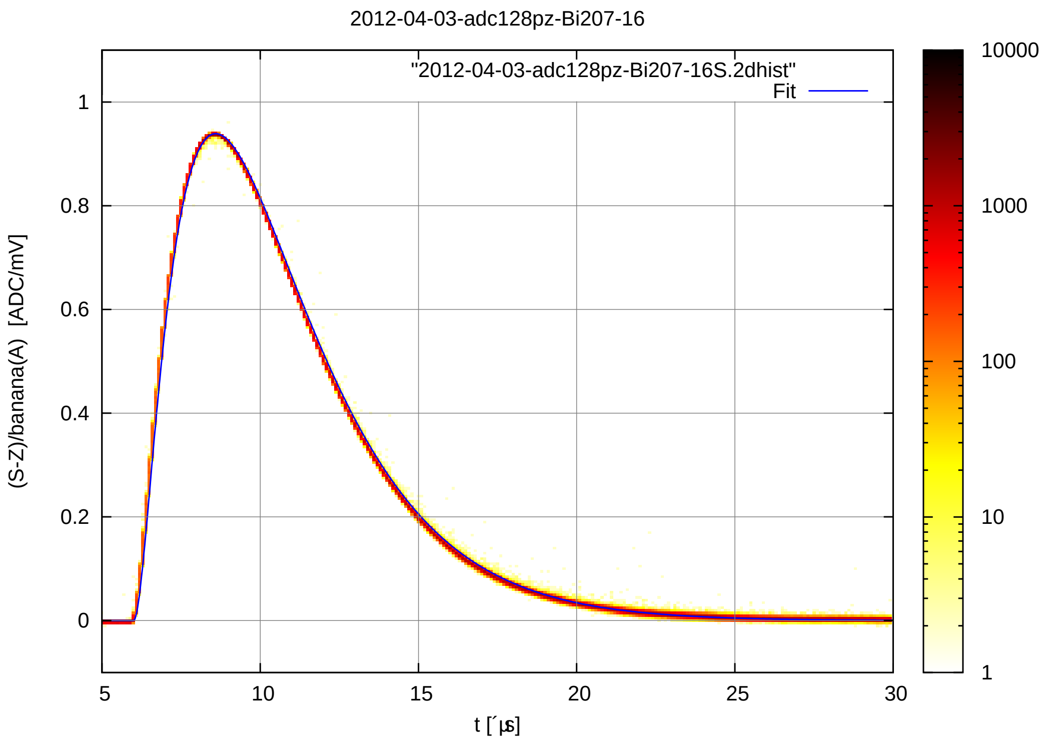

Figure 8: Samples and pulse shape fit. The color is the sample density. A scatterplot with this many dots would kill the printer.

the same time constants for high gain

$$C_2 = 200\,\mathrm{pF}, \qquad R_2 = 10\,\mathrm{k}\Omega, \tag{39}$$

$$C_1 = 10\,\mathrm{nF}, \tag{40}$$

$$R_1 = 220\,\Omega \quad \text{on the preamplifier board,} \tag{41}$$

$$C_0 = 220\,\mathrm{nF}, \qquad R_0 = 10\,\mathrm{k}\Omega, \tag{42}$$

$$\tau = 2.2\,\mu\mathrm{s}, \qquad \tau_r = 105\,\mu\mathrm{s}, \qquad \tau_0 = 2300\,\mu\mathrm{s}. \tag{43}$$

Two preamplifier were modified for larger dynamic range, with $C_r = 10\,\mathrm{pF}$, $R_r = 10\,\mathrm{M}\Omega$, and $C_r = 3.3\,\mathrm{pF}$, $R_r = 33\,\mathrm{M}\Omega$. The third preamplifier connected to a pole-zero enabled shaper was kept with the standard high-gain configuration $C_r = 1\,\mathrm{pF}$, $R_r = 100\,\mathrm{M}\Omega$.

Hamamatsu photodiodes size $1\,\mathrm{cm}^2$ were connected to all five the preamplifier inputs and biased to $U_{\mathrm{bias}} = 67\,\mathrm{V}$. The ^{207}Bi conversion electron source was mounted in front of the photodiodes, at a distance of about 1 cm.

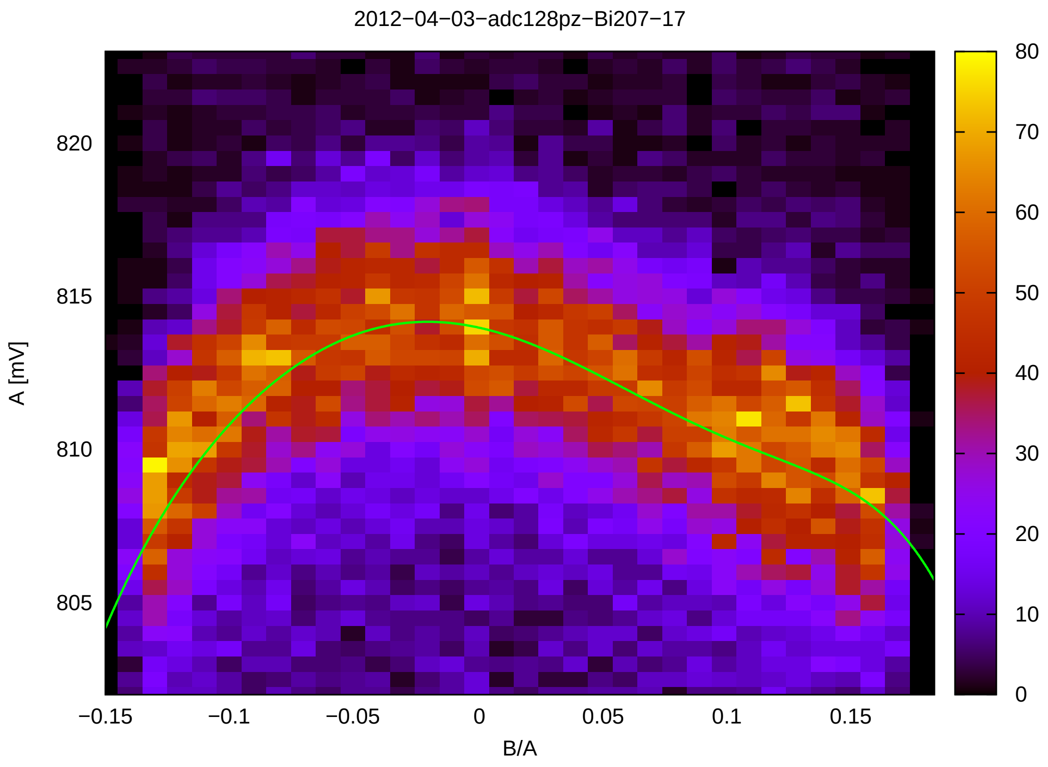

Figure 9: Banana plot of the 976 keV line, with the $b = 0.970\,\%$ coefficients. Also shown the polynomial banana fit to the pulse shape analysis model.

5.1 Preamplifier stability

The $C_r = 10\,\text{pF}$ preamplifier is populated with flight UHF transistors. This was the first test we did with this preamplifier design with such a large feedback capacitor. No instabilities were observed. With the feedback loop closed right after the common base stage instabilities were not expected, but still, it good to see it working. A proper analysis of the phase margin is still pending.

5.2 Low noise analysis, moderate banana coefficients

An attempt was made to run with the tuned, high-resolution coefficients discussed before:

```
0.00970 0.830 9.159 -0.084 0.93 [0, 0, -1, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
0.00970 0.830 9.159 -0.047 1.04 [3, -1, -3, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]
```

That resulted in multiple triggers per pulse, observable by extra peaks in the spectra at lower energies. The untuned coefficients from the first line

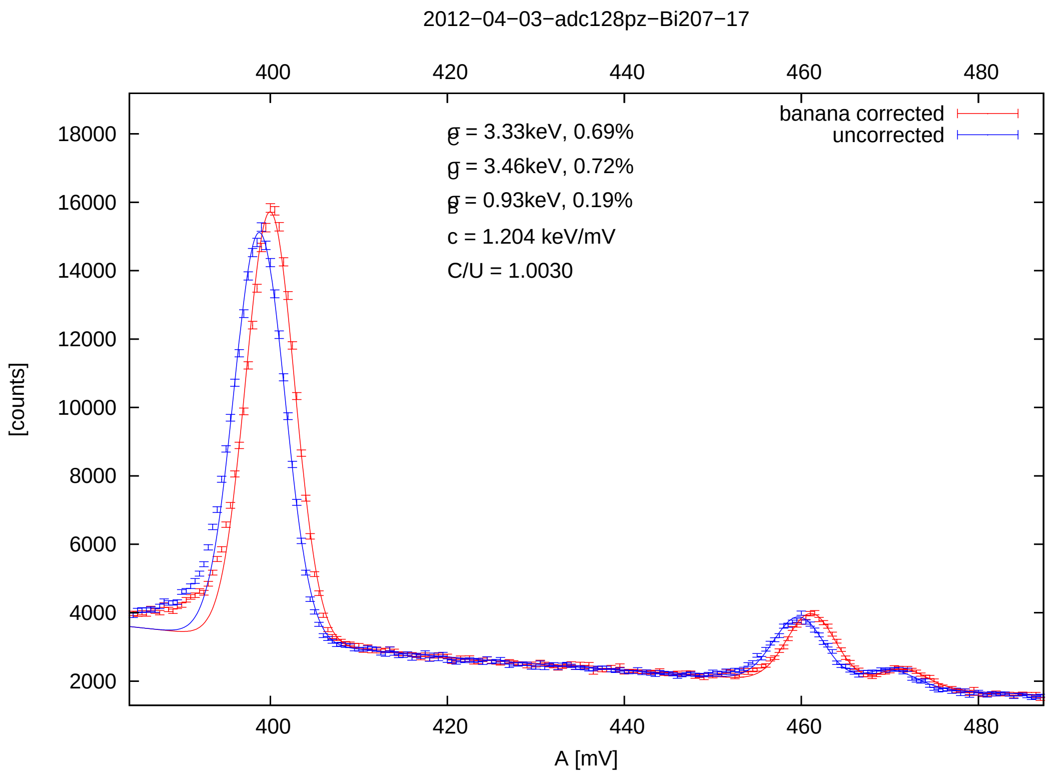

Figure 10: Resolution fits to the 482 keV electron lines. Data and fits, for the uncorrected and banana-corrected spectra. (The σ symbols did not print.)

were used instead. These coefficients yield an exceptional high amplitude value, with correspondingly good noise parameter, but the banana error is expected to be up to ±0.5 %.

5.2.1 Pulse shape fit

A samples run was taken with a high trigger threshold, so that only 976 keV electron hits were collected. Preliminary banana and phase fits were performed, and the the ADC samples were normalized to unit amplitude and proper phase. The pulse shape model from section 2.5 was fitted to the data. The results were

$$\begin{aligned} \tau &= 2.23077 \pm 0.00017\,\mu\text{s}, && (44)\\ \tau_r &= 2.4{\cdot}10^8 \pm 1.5{\cdot}10^{11}\,\mu\text{s}, && (45)\\ \tau_f &= 0.3117 \pm 0.0004\,\mu\text{s}. && (46)\\ & && (47) \end{aligned}$$

The expected pole at $\tau_r = 2300\,\mu$s is missing.

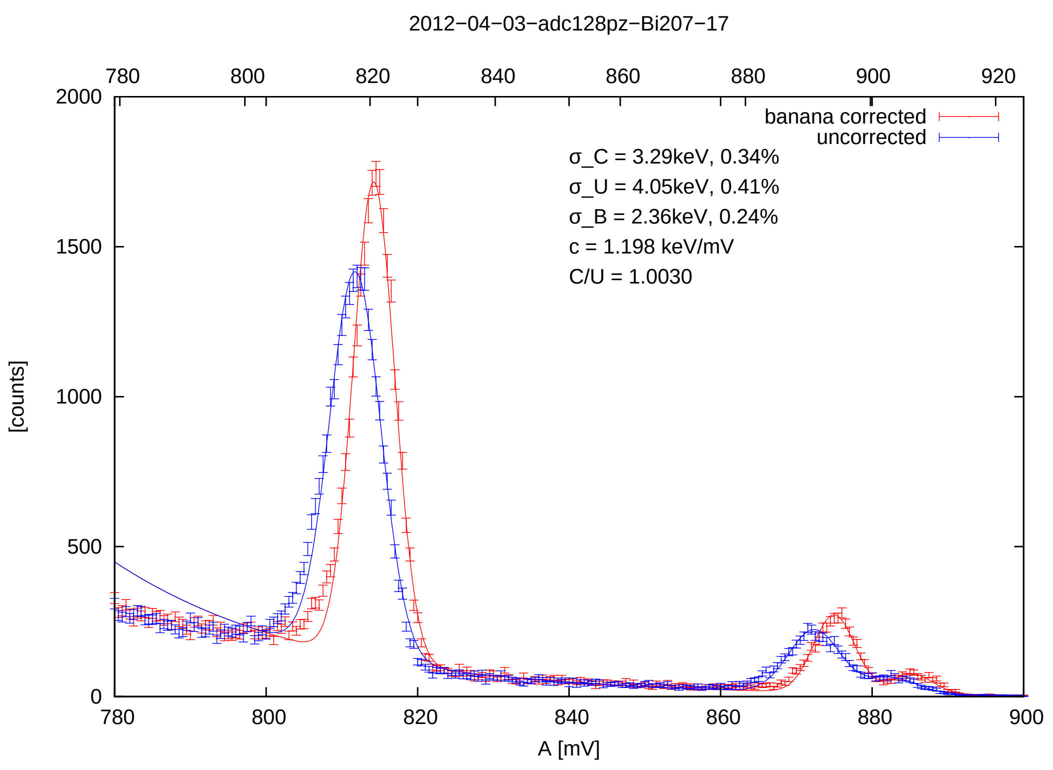

Figure 11: Resolution fits to the 976 keV electron lines. Data and fits, for the uncorrected and banana-corrected spectra.

A polynomial banana and phase fit was then performed on the model pulse shape, yielding a correction term for $f = B/A$

$$C(f) = 1 - 0.021f - 0.481f^2 + 1.943f^3 + 9.219f^4 - 8.390f^5 + 277f^6. \quad (48)$$

The corrected amplitude is then

$$A_c = \frac{A}{C\left(\frac{B}{A}\right)}. \quad (49)$$

The phase fit results were

$$\varphi = 1.395 - 2.964f + 0.387f^2 - 17.2f^3 + 5.296f^4 + 171.9f^5. \quad (50)$$

The constant term is a result of the time normalization of the scripts. It can be omitted.

The ADC samples were normalized again with these corrections, and a new pulse shape model fit performed, yielding about the same results:

$$\tau = 2.22938 \pm 0.00017\,\mu\text{s}, \quad (51)$$

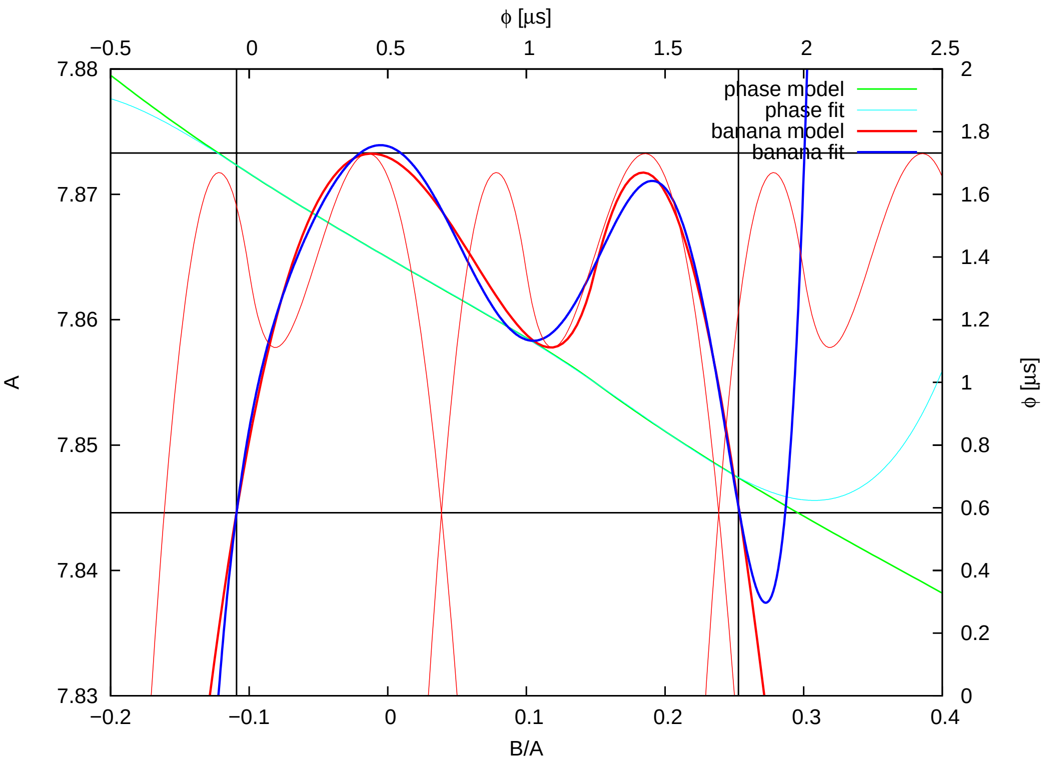

Figure 12: Banana fit for $b = 0.385\,\%$ coefficients. Thin red is the amplitude vs phase $A(\varphi)$ (left vs top axis), for three ADC clock periods. Thick red is the same data vs the phase parameter $A(\frac{B}{A})$ (bottom axis) and the polynomial of degree 7 fitted in the range $[-0.116 \le \frac{B}{A} \le 0.256]$ in blue. The black bars frame the range of the banana that need to be covered. In green is phase vs phase parameter $\varphi(\frac{B}{A})$ (right vs bottom axis), with a polynomial fit in light blue. The coefficients are
$\tau = 2.229\,\mu$s, $\tau_r = 240s$, $\tau_f = 0.3177\,\mu$s,
a_i=$[0, 0, 1, 2, 3, 2, 2, 1, 1, -3, -3, -3, -3, 0, 0, 0]$,
b_i=$[0, 1, 1, 2, 2, 0, -1, -3, -2, 0, 0, 0, 0, 0, 0, 0]$,
$b = 0.385\,\%$, $A_{\max} = 7.873$, $n = 0.98$, $p = -8.9\,\%$.

$$\tau_r = 2.4{\cdot}10^8 \pm 1.4{\cdot}10^{11}\,\mu\text{s}, \tag{52}$$
$$\tau_f = 0.3177 \pm 0.0004\,\mu\text{s}. \tag{53}$$
$$\tag{54}$$

The normalized sample coordinates are

$$y = (s_i - s_{15})C(\frac{B}{A})/A \tag{55}$$

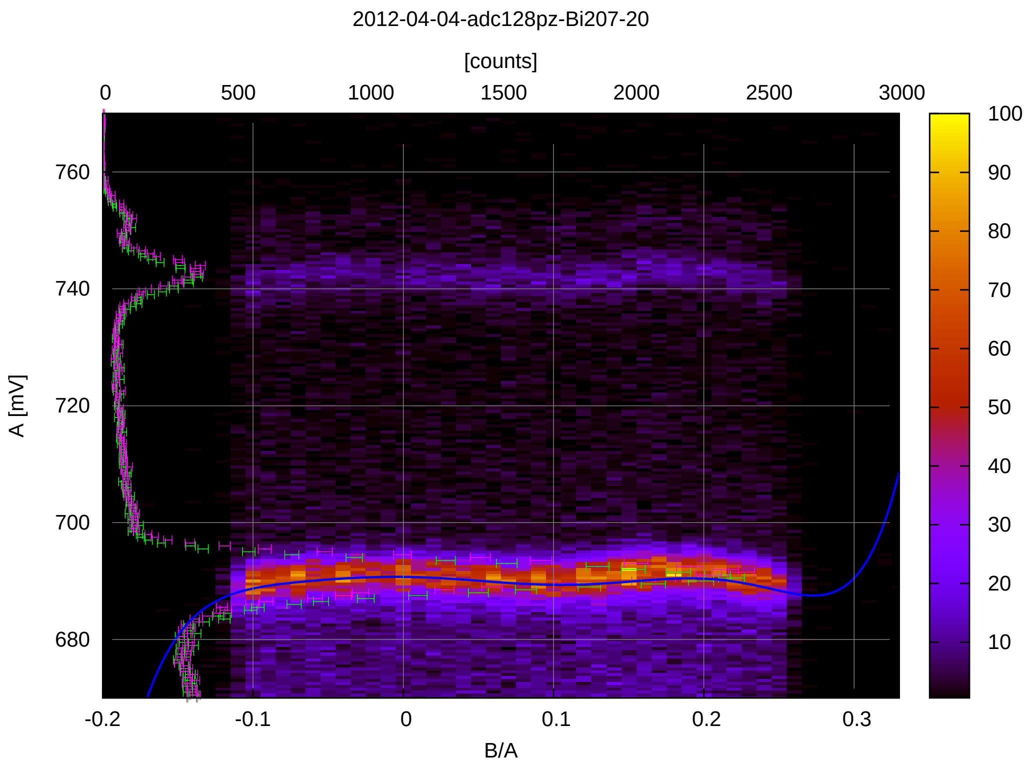

Figure 13: Banana plot for $b = 0.385\,\%$ coefficients. With the spectra plotted x vs y-axis, green: uncorrected, pink: banana-corrected. In blue, the banana correction function, derived from the pulse shape model.

$$x = 15 - i + \varphi(\frac{B}{A}) \qquad (56)$$

Fig. 8 shows the sample density and the pulse model fit.

5.2.2 Banana plot

Fig. 9 show how well the model banana correction works. The green line was derived from the pulse shape model only with some approximate amplitude factor applied.

5.2.3 Energy resolution

The spectra were analyzed both uncorrected and with banana-correction applied. Figs. 10 and 11 show fits to the electron line spectra in at two energies, 482 keV and 976 keV. The resolution of the corrected peaks is 3.3 keV in both cases. The banana error adds 0.19 % and 0.24 % to the resolution in the uncorrected spectra. The banana connection moves the peaks by +0.3 % up.

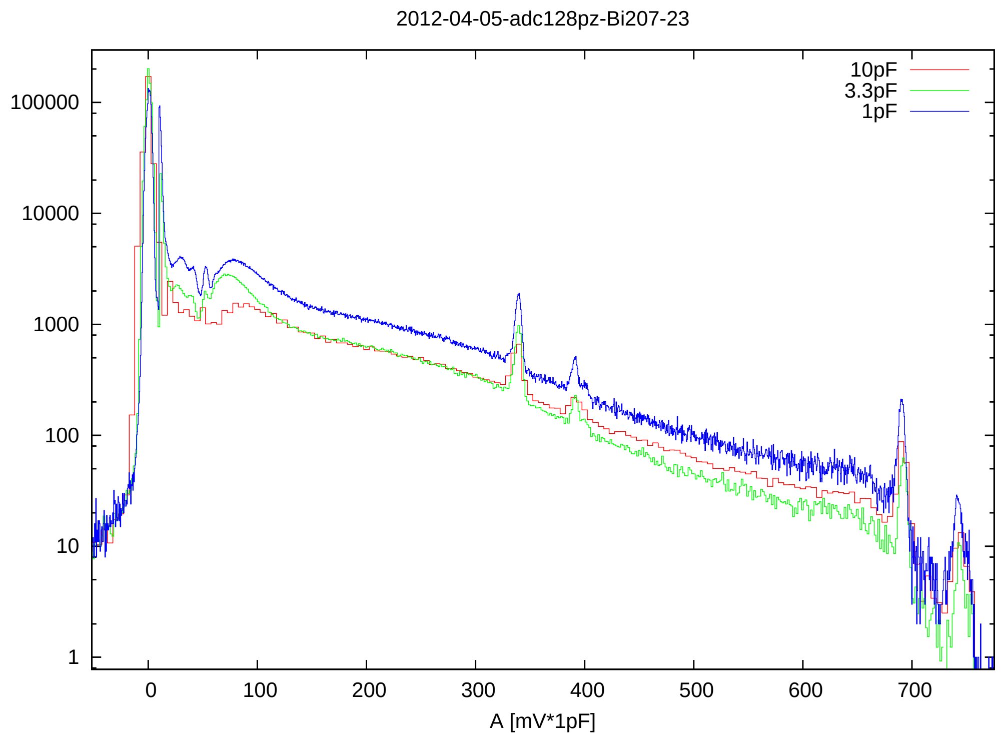

Figure 14: ^{207}Bi-spectra for all gains. The $C_r = 3.3\,\mathrm{pF}$ and $C_r = 10\,\mathrm{pF}$ spectra were scaled to match. The spectra were *not* banana-corrected.

The calibration of the connected spectra is 1.204 mV/keV and 1.198 mV/keV. The difference could be energy loss differences in the air and detector windows of about 3 keV

5.3 Low noise, good banana, coefficients

Next, I used the pulse shape model as fitted to the measured pulse samples to repeat the search for optimal coefficients. The best banana with noise parameter below 1 is

```
0.00385 0.690 7.873 -0.089 0.98 [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 3, 2, 1, 0, 0]
```

The model analysis was performed and a banana fitted to the model. This shape required an extra order in the polynomial (Fig. 12)

$$\begin{aligned} C(f) &= 1 - 0.003928f - 0.3721f^2 + 0.2726f^3 \\ &\quad +22.70f^4 + 36.51f^5 - 964.3f^6 + 2091f^7. \end{aligned} \tag{57}$$

Fig. 13 shows the measured 976 keV banana.

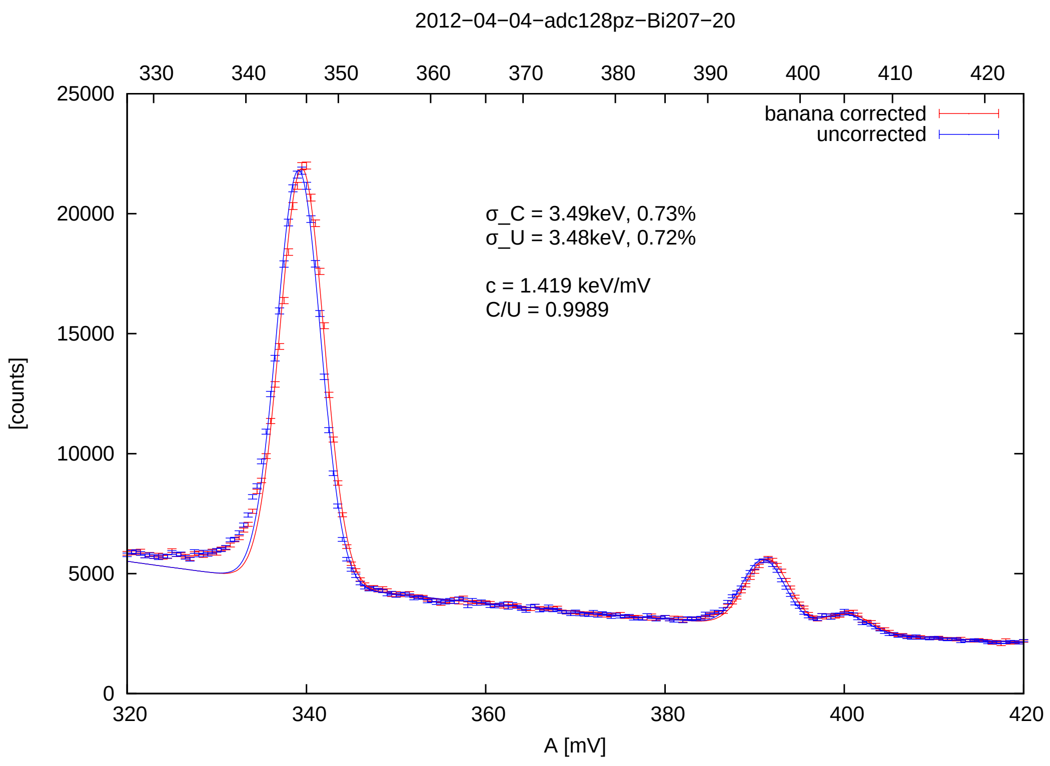

Figure 15: Resolution fits to the 482 keV electron lines. Data and fits, for the uncorrected and banana-corrected spectra.

5.3.1 Energy resolution

The spectra for all three channels are shown in Fig. 14. The trigger thresholds were agressivly lowered, to 1.8 mV, 3.5 mV, and 10 mV.

The high events rates just above threshold are not caused by noise but by the 10.6 keV X-ray line of the Bismuth source.

Fig. 15 shows the resolution fits. The banana correction did not improve the resolution. Overall, the resolution is worse than with the low-noise coefficients, by about 5 %

$$\frac{\sigma_{b0.385}}{\sigma_{b0.970}} = \frac{3.49\,\text{keV}}{3.33\,\text{keV}} = 1.048. \tag{58}$$

This agrees nicely with the rations of the noise parameters of the coefficients

$$\frac{n_{b0.385}}{n_{b0.970}} = \frac{0.98}{0.93} = 1.05. \tag{59}$$

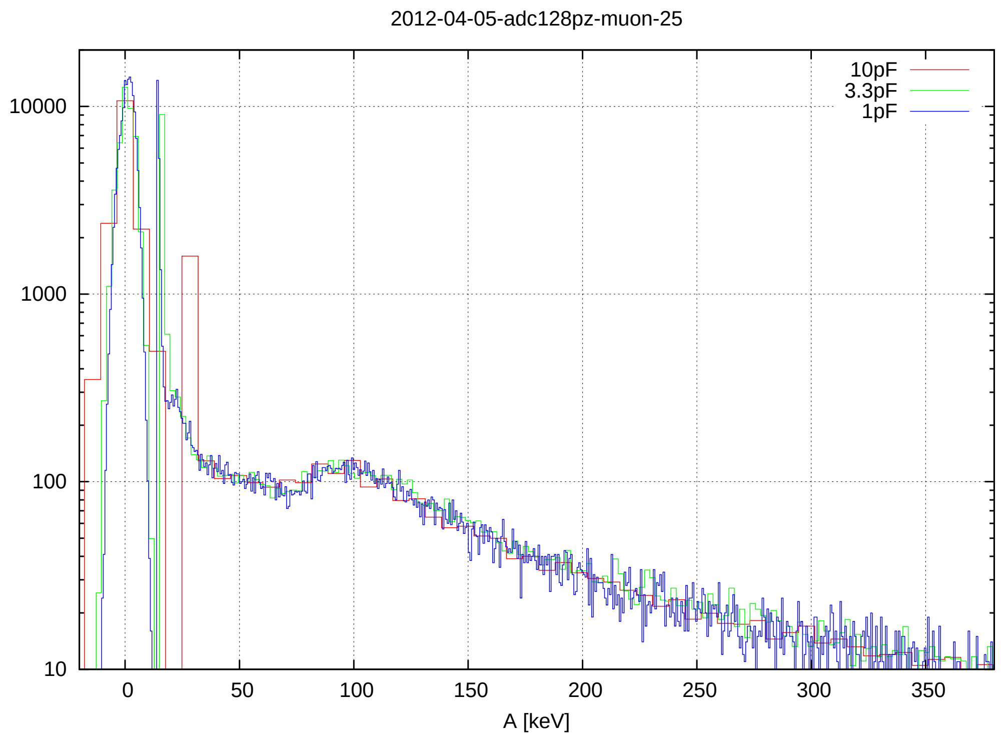

Figure 16: Cosmic muon spectra.

5.3.2 Trigger threshold

The spectra shown in Fig. 16 were accumulated over five days with cosmic muons. The trigger thresholds were adjusted to just acceptable noise trigger rate at 14 keV, 16 keV, and 28 keV respectively for the three different preamplifier gains.

6 Conclusion

The flight DIRENA implementation can deliver pulse heights with less than 1 % banana error. The techniques and scripts developed for this study supply the tools to derive coefficients suitable for flight.

The recovery speed can be traded for noise. In light of these tradeoffs it may be advisable to keep the coefficients configurable even with the final flight FPGA. On the other hand, the achieved results are better than required for all parameters, and the tradeoffs yield only marginal improvements for one or the other parameter.

7 Python script

Started as a program, the scripts computes and outputs the curves shown in the figures. Importing as a module allows to run the function `exsearch()` to find all solutions with $b < 1\,\%$. The parameters and coefficients were edited in the source.

```
#! /usr/bin/python
# -*- encoding: utf-8 -*-
from math import exp, sqrt
import numpy, sys

# from 2011-06-01-Bi207-14-fit.gpt

T=  1.02271409701796 # µs
TR= 110.784997349802 # µs
TP= 0.363295701847157 # µs
t0= 8.05429549655105 # µs
A=  2.38912672508397

# slow shapers:

T = 2.2 # µs
TR= 330. # µs
t0 = 0

TR=1000 # with pole-zero correction

def shaper(t):
  if t<=0:
    return 0.0
  s1 = -exp(-t/TR) * TR / ( (TR-TP)*(TR-T)**2 )
  s2 = -exp(-t/T) * (TP*TR-T**2) / ( (TP-T)**2 * (TR-T)**2 )
  s3 = exp(-t/TP) * TP / ( (TP-T)**2 * (TR-TP) )
  s4 = -t*exp(-t/T) / ( T*TP*TR - T**2*(TR+TP) + T**3 )

  #s1 = -TR*exp(-(t/TR))/(TR**3+(-TP-2*T)*TR**2+(2*T*TP+T**2)*TR-T**2*TP)
  #s2 = -exp(-(t/T))*(T*TP*TR-T**3)/(T*((TP**2-2*T*TP+T**2)*TR**2
  #             +(-2*T*TP**2+4*T**2*TP-2*T**3)*TR+T**2*TP**2
  #             -2*T**3*TP+T**4))
  #s3 = TP*exp(-(t/TP))/((TP**2-2*T*TP+T**2)*TR-TP**3+2*T*TP**2-T**2*TP)
  #s4 = -t*exp(-(t/T))/(T*((TP-T)*TR-T*TP+T**2))
  s = s1+s2+s3+s4
  return A*T*TR*s

# 0.0055
fa=[  0,  0,  0, -3, -3, -3, -1,  1,  1,  2,  3,  2,  2,  1,  0,  0, ]
fb=[  0,  0,  0,  0,  0,  0,  0, -2, -3,  -2,  0,  3,  2,  1,  1,  0, ]
```

```
# March runs
fa=[-1, -1, -1, -1, -2, -2, -2, 0, 1, 3, 3, 2, 1, 0, 0, 0, ]
fb=[0, 0, 0, 0, 0, 0, 0, -3, -2, 0, 1, 2, 1, 1, 0, 0, ]

# 0.0035
fa = [ 0,  0,  0, -1, -2, -3, -2,  1,  0,  3,  2,  2,  1,  0,  0,  0, ]
fb = [ 0,  0,  0,  0,  0,  0,  0, -2, -3, -1,  0,  2,  2,  1,  1,  0, ]

# 0.0030
fa = [-0, -0, -0, -0, -3, -3, -3,  1,  0,  3,  2,  1,  2,  0,  0,  0, ]

# TR=330. 0.00330
fa = [0, 0, 0, -1, -3, -3, -3, 1, 0, 3, 3, 0, 0, 3, 0, 0]
# 0.00378
fa = [0, 0, 0, 0, -2, -3, -3, 1, 0, 2, 3, 3, 0, -1, 0, 0]
# 0.00305
fa = [ 0,  0,  0,  0, -3, -3, -3,  1,  0,  3,  2,  1,  3,  0, -1,  0, ]

# TR=10000
# 0.00311
fa = [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 3, 1, 0, 1, 0]

# 0.00287
fa = [0, 0, 0, -2, -3, -3, -3, 1, 1, 1, 3, 2, 1, 1, 1, 0]

# 0.00385 0.660 6.259 -0.051
fa = [0, 1, 2, -2, -3, -3, -3, 1, 0, 2, 3, 3, 1, -1, -1, 0]

# TR = 1000
# 0.00446 0.680 7.791 -0.088 0.99
fa = [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 3, 2, 1, 0, 0]

# 0.00995 0.730 5.434 -0.046 1.16
fa = [0, 2, 2, -2, -3, -3, -3, 1, 0, 2, 2, 3, 1, -1, -1, 0]

# 0.00970 0.830 9.159 -0.080 0.96

fa = [3, -1, -3, -3, -3, -3, -3, 1, 1, 3, 3, 3, 2, 1, -1, 0]

# 0.00360 0.670 6.110 -0.052 1.11
fa = [0, 1, 3, -3, -3, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]
fa = [0, 1, 2, -2, -3, -3, -3, 1, 0, 2, 3, 3, 0, 0, -1, 0]

# 2012-03-30-hetdm128-Bi207-12
fa = [-0, -0, -0, -0, -3, -3, -3,  1,  0,  3,  2,  1,  2,  0,  0,  0, ]
fb = [ 0,  0,  0,  0,  0,  0,  0, -2, -3, -1,  0,  2,  2,  1,  1,  0, ]

# 2012-04-03-adc128pz-Bi207-17.dat
```

```
fa=[0, -1, 1, 2, 3, 3, 3, 1, 1, -3, -3, -3, -3, -1, 0, 0]
fb=[0, 1, 1, 2, 2, 0, -1, -3, -2, 0, 0, 0, 0, 0, 0, 0]
fa.reverse()
fb.reverse()
T               = 2.23077     #     +/- 0.0001689    (0.00757%)
TR              = 2.40239e+08 #     +/- 1.479e+11    (6.156e+04%)
TP              = 0.311688    #     +/- 0.0003724    (0.1195%)
#G               = 2.58376     #     +/- 0.0001471    (0.005692%)
#X               = 5.996       #     +/- 0.0002728    (0.00455%)
T               = 2.22938     #     +/- 0.0001659    (0.007442%)
TR              = 2.40239e+08 #     +/- 1.444e+11    (6.011e+04%)
TP              = 0.317707    #     +/- 0.0003644    (0.1147%)

# 2012-04-04-adc128pz-Bi207-19
# 0.00391 -0.380 7.290 -0.143 1.06
fa = [0, 0, 0, -3, -3, -3, -3, 1, 1, 1, 1, 3, 3, 1, 1, 0]

# 2012-04-04-adc128pz-Bi207-20
# 0.00385 0.690 7.873 -0.089 0.98
fa = [0, 0, 0, -3, -3, -3, -3, 1, 1, 2, 2, 3, 2, 1, 0, 0]

if sum(fa) or sum(fb):
  raise ValueError("fa must be normalised to zero")

def AB(fab, phase):
  s = numpy.array([shaper(t-8+phase) for t in range(16)])
  return [sum(s*a) for a in fab]

def width(a,m):
  w1=0
  w2=len(a)
  for i,aa in enumerate(a):
    if aa>=m:
      w2 = i;
      if not w1:
        w1 = i
    elif w1:
      break
  return w2-w1;

def banana(fa):
  a = numpy.array([AB((fa,), p/100.-1)[0] for p in range(401)])
  m = a.max()
  m1 = 0
  m2 = m
  while m2-m1 > 0.0001*m:
    mm = (m1+m2)/2.0;
    if width(a, mm) > 100:
      m1 = mm
    else:
      m2 = mm
  return 1-m1/m

def poffset(fa):
  b = banana(fa)
  a = numpy.array([AB((fa,), p/100.-1)[0] for p in range(401)])
  m = a.max()*(1-b)
  for p,aa in enumerate(a):
    if aa>=m:
      pp = p/100.-1
      return pp, a.max(), AB((fa,), 15+pp)[0]/a.max()

def ismin(fa, i):
  return False
  return fa[i]<fa[i-1] and fa[i]<fa[i+1]

def findfa(fa):
  fam = None
  bmin = banana(fa)
  fan = [a for a in fa]
  for i in range(7,15):
    fan[i] = fa[i]+1
    if fan[i]<=3 and not ismin(fan,i-1) and (i>=14 or not ismin(fan,i+1)):
      b = banana(fan)
      if b<bmin:
        bmin = b
        fam = [a for a in fan]
    fan[i] = fa[i]-1
    if fan[i]>=-3 and not ismin(fan,i):
      fan[i] = fa[i]-1
      b = banana(fan)
      if b<bmin:
        bmin = b
        fam = [a for a in fan]
    fan[i] = fa[i]
  return bmin, fam

def noiseterm(fa):
  return sqrt(sum([a*a for a in fa]))

fas = []

def exsearch():
  fam = None
  fa = [0]*16
  fa[6]=-3
  fa[5]=-3
  fa[4]=-3
  bmin = 1
  r = [0,1,2,3]
  for fa[14] in [0,-1,1]:
    for fa[13] in [-1,0,1]:
      for fa[12] in r:
        for fa[11] in r:
          for fa[10] in r:
            for fa[9] in [1,2,3]:
              for fa[8] in [0,1]:
                for fa[7] in [1]:
                  fa[0:7] = [0,0,0,0,-3,-3,-3]
                  s = sum(fa)
                  if s<0:
                    fa[4] = -3-s
                  else:
                    fa[3] = -s
                  b = banana(fa)
                  if b<bmin:
                    bmin = b
                    fam = [a for a in fa]
                  if b<0.01:
                    p, m, t = poffset(fa)
                    fas.append((b, p, m, t, noiseterm(fa)/m, [a for a in fa]))
                    if b<0.0051:
                      sys.stderr.write("%.5f %.3f %.3f %.3f %.2f %s\n" % fas[-1])
  return bmin, fam

fan = [a for a in fa]
while fan:
  b = banana(fan)
  p, m, t = poffset(fan)
  n = noiseterm(fan)/m
  sys.stderr.write("%.5f %.3f %.3f %.3f %.2f %s\n" % (b, p, m, t, n, fan))
  b,fan = findfa(fan)

if __name__ == '__main__':
  p = poffset(fa)[0]
  ss = 0
  for t in range(5501):
    tt = t/100. - 10
    i = int(tt+8-p)
    if i<0 or i>15:
      i=15
    a,b = AB((fa,fb), tt)
    s = shaper(tt)
    print tt, s, a,b, fa[i], fb[i], (s-ss)*100
    ss = s
```