Dorn Signal Processing

Stephan I. Böttcher

dorn-shaper.tex 8184 2021-03-12 20:33:00Z stephan

Contents

List of Figures

List of Tables

1 DORN

The instrument DORN shall fly on the Chinese Mission Chang'E 6 to the moon to measure α particles from the decay of Radon. There will be thin Silicon detectors and a plastic anticoincidence read out via photodiodes. There shall be eight detector units with three detector channels each. The detector signals are fast charge pulses

$$i(t) = q\delta(t). \tag{1}$$

The charge is proportional to the energy deposited in the α-detectors

$$q = e_0 \frac{E_{\text{dep}}}{3.6\,\text{eV}}. \tag{2}$$

In case of the anticoincidence detector the charge is that of the released photo electrons.

The trigger rate in the relevant energy range above 5 MeV is expected to be low.

This document describes a proposal to readout each detector unit by one ADC with a sample cadence of 3 μs per channel. Trigger and pulse height determination will be performed digitally in an FPGA.

Heritage for this proposal are balloon experiments, where we employ one ADC per channel with 3 MSPS, and the instrument HET/EPT on Solar Orbiter, also with one ADC per channel, and 1 μs sample cadence. On Solar Orbiter we expect high trigger rates, with fast shaping times and more digital filtering and fully parallel digital processing. The employed flight part ADC128S102 has eight multiplexed inputs. On Solar Orbiter seven of the inputs remain unused.

For DORN we propose to use three inputs for detector input and five channels are available for housekeeping. To achieve the necessary resolution we add two more poles to the analog filter in form of a Sallen-Key low pass. The digital processing will be pipelined though a single processor chain for all detector units. Heritage instruments attempt to trigger just above the noise. The DORN trigger thresholds are way above the noise. The trigger can be performed before doing the digital filter calculations. That saves power.

The output will be timestamped pulse height records for all channels of a detector units where at least one channel issued a trigger.

2 Analog Signal Processing

The detector signals Eq. (1) are going through a *Charge Sensitive Amplifier* (CSA), a shaper, a Sallen-Key filter and an ADC input filter.

In this section we develop the transfer function of this signal chain. Since the input is a δ-function, the pulse response functions directly represent the signal forms.

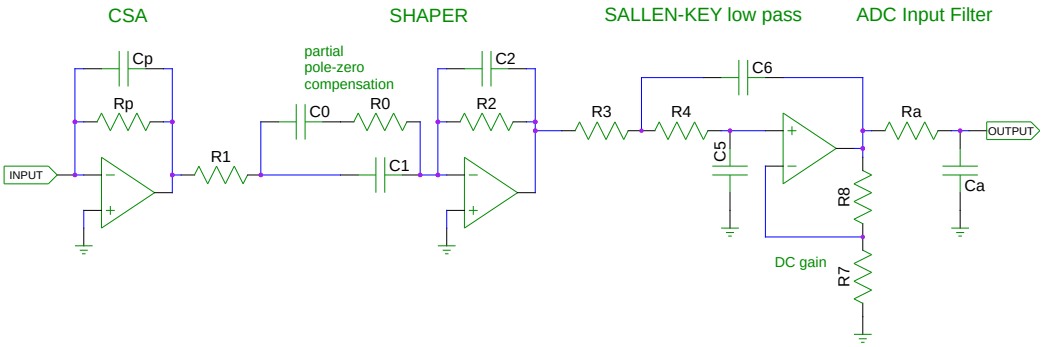

Figure 1: Schematic diagram of the analog signal processing.

2.1 Charge Sensitive Amplifier

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

$$h_P(t) = -\frac{1}{C_P}\Theta(t)\,\exp(-t/\tau_P). \tag{3}$$

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

$$H_P(s) = \mathcal{L}\left\{h_P(t)\right\} = -\frac{1}{C_P}\,\frac{\tau_P}{1+s\tau_P}. \tag{4}$$

2.2 Shaper

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

$$g_S = \frac{V_{\mathrm{OUT}}}{V_{\mathrm{IN}}} = -\frac{Z_2}{Z_1} \tag{5}$$

Z_1 is the impedance of the input branch, a resistor R_1 and a capacitor C_1 in series, with time constant $\tau = R_1 C_1 = 2.7\,\mu\text{s}$

$$Z_1 = R_1 + \frac{1}{j\omega C_1} = R_1 \frac{1 + j\omega\tau}{j\omega\tau}. \tag{6}$$

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 $\tau = R_2 C_2 = 2.7\,\mu\text{s}$

$$Z_2 = \frac{\frac{R_2}{j\omega C_2}}{R_2 + \frac{1}{j\omega C_2}} = \frac{R_2}{1 + j\omega\tau} \tag{7}$$

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

$$H_S(s) = -\frac{Z_2}{Z_1} = -\frac{as\tau}{(1 + s\tau)^2} \tag{8}$$

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

2.3 Pole-Zero Compensation

The pole of the CSA causes an undesired undershoot in the pulse response. 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:

$$Z_1 = R_1 + \frac{R_0 \frac{1}{j\omega C_1}}{R_0 + \frac{1}{j\omega C_1}} = R_1 \frac{\tau_P}{\tau} \frac{1 + j\omega\tau}{1 + j\omega\tau_P}, \tag{9}$$

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

$$H_S(s) = -\frac{Z_2}{Z_1} = -\frac{a\tau}{\tau_P} \frac{1 + s\tau_P}{(1 + s\tau)^2}. \tag{10}$$

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

$$H_P(s) H_S(s) = \frac{1}{C_P} \frac{a\tau}{(1 + s\tau)^2}. \tag{11}$$

The resistor R_0 introduces a DC-gain to the circuit. The preamplifier outputs have a DC level at about $V_{\text{GS}} = -250\,\text{mV}$, that is the gate-source voltage of the JFET at the chosen bias current. The resulting DC offset will cut into the

dynamic range of the channel. 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 $C_1 = 2.7\,\mathrm{nF}$ this may not be negligible.

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

$$\begin{aligned} Z_1 &= R_1 + \frac{\frac{1}{j\omega C_1}\left(R_0 + \frac{1}{j\omega C_0}\right)}{\frac{1}{j\omega C_1} + R_0 + \frac{1}{j\omega C_0}} && (12)\\ &= R_1 \frac{1 + s(\tau_1 + \tau_{00}) + s^2\tau_1\tau_P}{(1 + s\tau_P)s\tau_1} && (13)\\ &= \frac{R_1}{s\tau_1}\frac{(1 + s\tau)(1 + s\tau_0)}{1 + s\tau_P} && (14) \end{aligned}$$

with

$$\begin{aligned} \tau_1 &= (C_0 + C_1)R_1 && (15)\\ \tau_{00} &= C_0R_0 && (16)\\ \tau_P &= C_1R_0\,C_0/(C_0 + C_1) && (17)\\ \tau\tau_0 &= \tau_1\tau_P = C_0C_1R_0R_1 && (18)\\ \tau + \tau_0 &= \tau_1 + \tau_{00} = R_1C_0 + R_1C_1 + R_0C_0. && (19) \end{aligned}$$

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 $(1 + s\tau_P)$ in the shaper transfer function

$$\begin{aligned} R_0 &= \frac{\tau_P}{C_1}\left(1 + \frac{C_1}{C_0}\right)^{-1}, && (20)\\ R_1 &= \frac{\tau}{C_1}\left(1 + \frac{\tau}{\tau_P} + \mathcal{O}\left\{\frac{\tau}{\tau_0}\right\}\right), && (21)\\ \tau_0 &= C_0R_0\left(1 + \frac{R_1}{R_0} + \mathcal{O}\left\{\frac{\tau^2}{\tau_P\tau_0}\right\}\right). && (22) \end{aligned}$$

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, we just need to replace τ_P by τ_0. With $C_1 = 2.7\,\mathrm{nF}$, $R_1 = \tau/C_1 = 1\,\mathrm{k\Omega}$, $R_1 = \tau_P/C_1 = 36\,\mathrm{k\Omega}$, and $C_0 = 220\,\mathrm{nF}$ the new pole with have the time constant $\tau_0 = 8\,\mathrm{ms}$.

2.4 Sallen-Key Filter

The transfer function of the Sallen-Key filter

$$H_F(s) = \frac{V_{\mathrm{OUT}}}{V_{\mathrm{IN}}}, \tag{23}$$

can be derived with Kirchhoff's circuit laws, solving this system of equations:

$$\begin{aligned}
V_3 &= Z_3 I_3, \\
V_4 &= Z_4 I_4, \\
V_5 &= Z_5 I_5, \\
V_6 &= Z_6 I_6, \\
V_7 &= Z_7 I_7, \\
V_8 &= Z_8 I_8, \\
V_{\mathrm{IN}} &= V_3 + V_4 + V_5, \\
V_5 &= V_7, \\
V_{\mathrm{OUT}} &= V_7 + V_8, \\
V_4 &= V_6 + V_8, \\
I_3 &= I_4 + I_6, \\
I_4 &= I_5, \\
I_7 &= I_8.
\end{aligned}$$

The solution is

$$H_F = \frac{Z_3 Z_4 Z_6 + Z_3 Z_4 Z_5}{Z_3 Z_4 Z_5 + Z_2 Z_3 Z_5 + Z_1 Z_3 Z_5 + Z_1 Z_2 Z_5 - Z_1 Z_4 Z_6}. \tag{24}$$

With $Z_3 = Z_4$ and $Z_5 = Z_6 = Z_3/(s\tau_F)$, $Z_8 = bZ_7$,

$$H_F(s) = \frac{b+1}{(1+s\tau_F)^2 - sb\tau_F}. \tag{25}$$

We will choose $\tau_F = \tau = 2.7\,\mu\mathrm{s}$.

2.5 ADC input filter

Between the filter output and ADC input there is a simple R-C filter. The resistor R_A is located close to the Sallen-Key Filter OPAMP, to limit its capacitive load to a minimum. The capacitor C_A 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 shall have a shorter time constant $\tau_A = R_A C_A = 1\,\mu$s. The output voltage range of the filter OPAMP is not much larger than the input voltage range of the ADC, a larger τ_A will cost dynamic range. The transfer function is that of a voltage divider

$$H_A(s) = \frac{Z_{C_A}}{R_A + Z_{C_A}} = \frac{\frac{1}{j\omega C_A}}{R_A + \frac{1}{j\omega C_A}} = \frac{1}{1 + s\tau_A} \tag{26}$$

2.6 Transfer Function

The complete analog transfer function is

$$\begin{aligned} H(s) &= H_P H_S H_F H_A \\ &= \frac{a(b+1)\tau\tau_0\, s}{C_P(1+s\tau_0)(1+s\tau)^2((1+s\tau)^2 - bs\tau)(1+s\tau_A)}. \end{aligned} \tag{27}$$

The inverse Laplace transforms were computed with `maxima`

$$h(t) = \mathcal{L}^{-1}(H(s)). \tag{28}$$

Fig. 2 shows how the functions look like.

2.7 Gain

The readout will be single gain per detector channel, covering energy deposits between a few and few tens of MeV. The proposal is to use high gain CSA with $C_P = 1\,$pF. The gain of the preamplifier is

$$g_P = -\frac{1}{C_P} = -\frac{1\,\mathrm{V}}{\mathrm{pC}}. \tag{29}$$

The output voltage range can support signals up to little more than 3 V.

$$E_{\mathrm{dep,max}} = 3\,\mathrm{V} \times \frac{1\,\mathrm{pC}}{\mathrm{V}} \times \frac{3.6\,\mathrm{eV}}{e_0} = 67\,\mathrm{MeV}. \tag{30}$$

The shaper shall have a gain of three with respect to the peak signal. That gives a signal of 130 mV/MeV and a range up to 22 MeV. To achieve unity gain the feedback gain factor needs to be $a = 2.7$. We need $a = 8.1$, with components $R_1 = 1\,\mathrm{k}\Omega$, $C_1 = 2.7\,$nF, $R_2 = 8.1\,\mathrm{k}\Omega$, $C_2 = 330\,$pF, and $R_0 = 36\,\mathrm{k}\Omega$.

The Sallen-Key filter shall provide unity gain, which requires a feedback resistor ratio $1/b = 2.2$, or a feedback gain of $(1 + R_7/R_8) = 1.45$. The

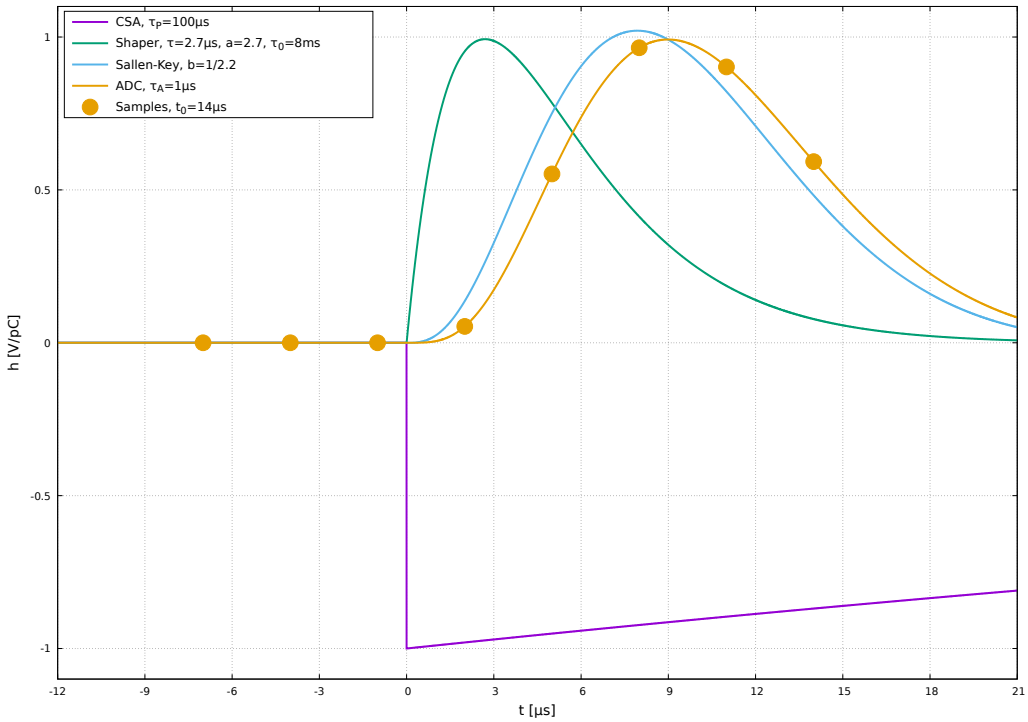

Figure 2: Pulse shapes. The tics on the x-axis are spaced according to the proposed sample cadence of $3\,\mu$s. A set of eight samples is shown that may be used to calculate the pulse height and phase.

component values shall be $R_3 = R_4 = 2.7\,\mathrm{k\Omega}$, $C_5 = C_6 = 1\,\mathrm{nF}$, $R_8 = 1\,\mathrm{k\Omega}$, and $R_7 = 2.2\,\mathrm{k\Omega}$.

The ADC filter reduces the signal height by a few percent. The component values are $R_A = 100\,\Omega$ and $C_A = 10\,\mathrm{nF}$.

The input voltage range of the ADC is $0\,\mathrm{V} \ldots 3.3\,\mathrm{V}$ with twelve bits resolution. That yields 6 keV per ADC channel.

2.8 Schematics

2.8.1 Dorn Channel

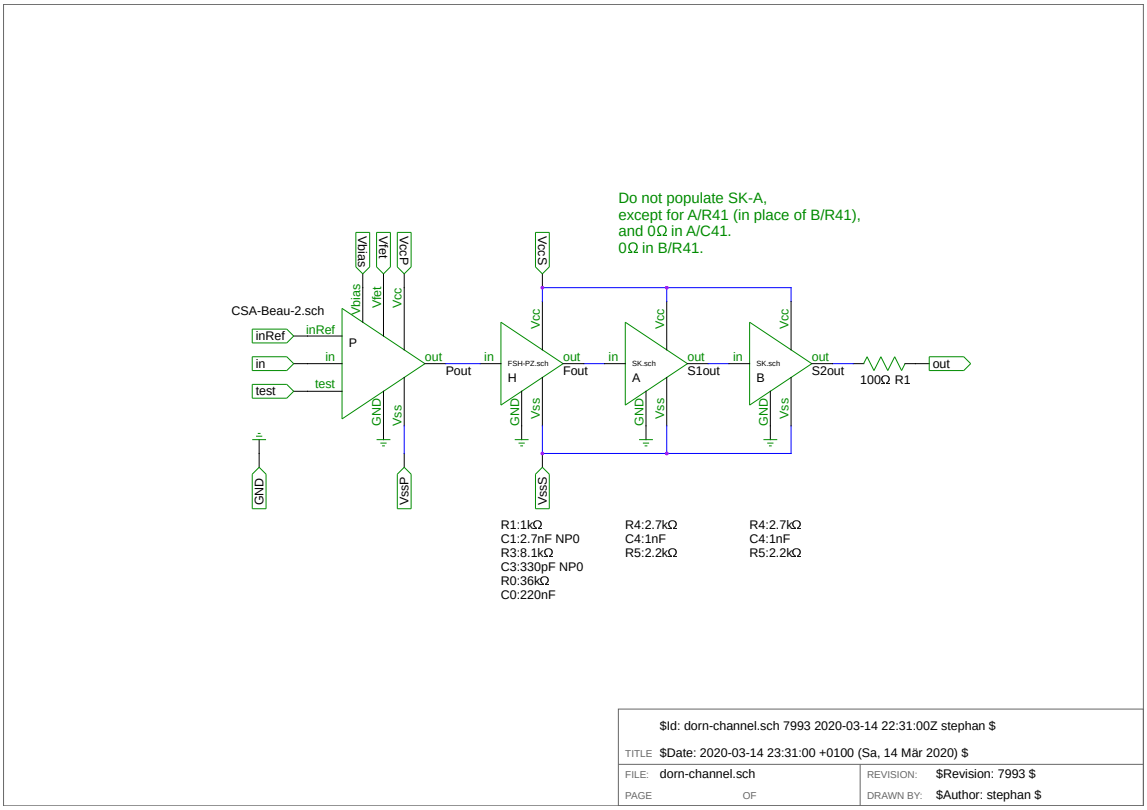

Figure 3: Dorn Channel. This page instantiates subschematics and provides values to parameters for the instances.

The first Sallen-Key filter `A` shall not be used. The second Sallen-Key filter `B` shall be implemented on the board which houses the ADCs, providing a low impedance driver for the ADC input and decoupling any clock-feed-through from the frontend. The output of the shaper `S` needs a series resistor close to the OPAMP output pin. Therefore, the input resistor A/R_{41} shall be populated instead of B/R_{41}. The footprints A/C_{41} and B/R_{41} shall be populated with a zero-Ω part or a wire.

2.8.2 Charge Sensitive Amplifier

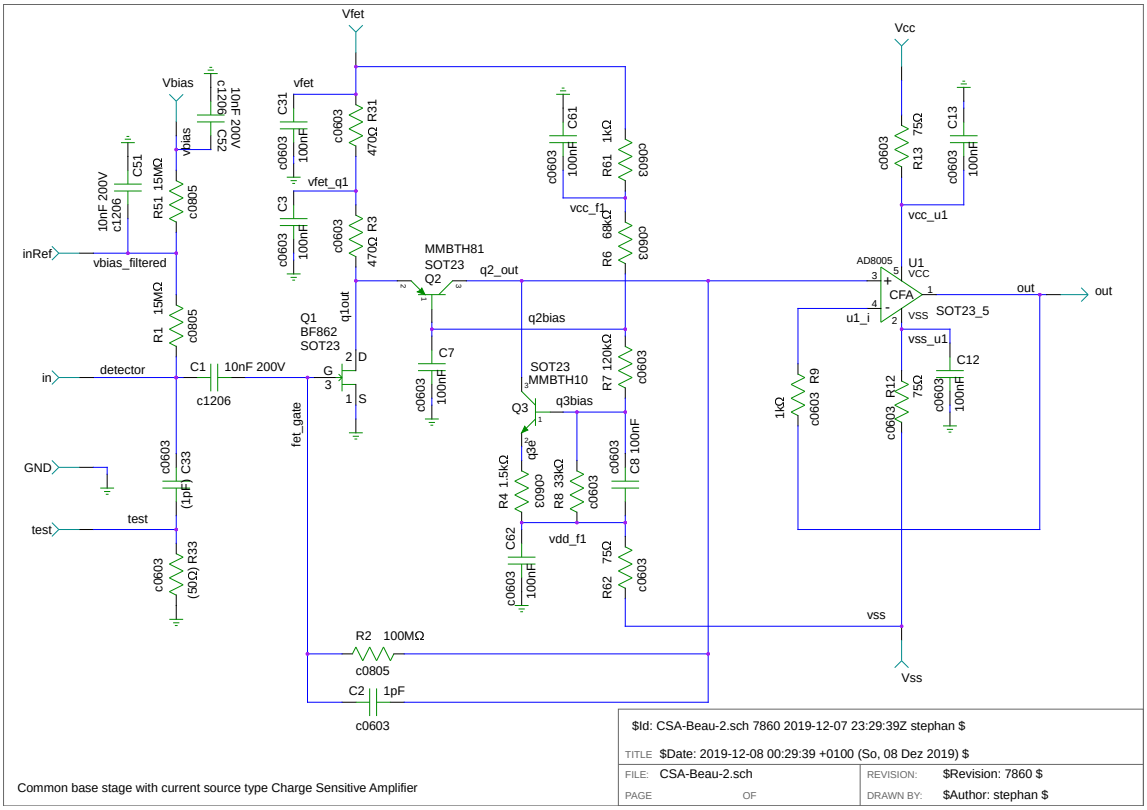

Figure 4: CSA schematics.

This schematics page has a lot of heritage. The design is based on work by Rudolf Beaujean. The OPAMP, as well as all OPAMPs in the shapers are AD8005 *Current Feedback Amplifiers* (CFA) which provide large bandwidth at low power and low noise.

2.8.3 Shaper

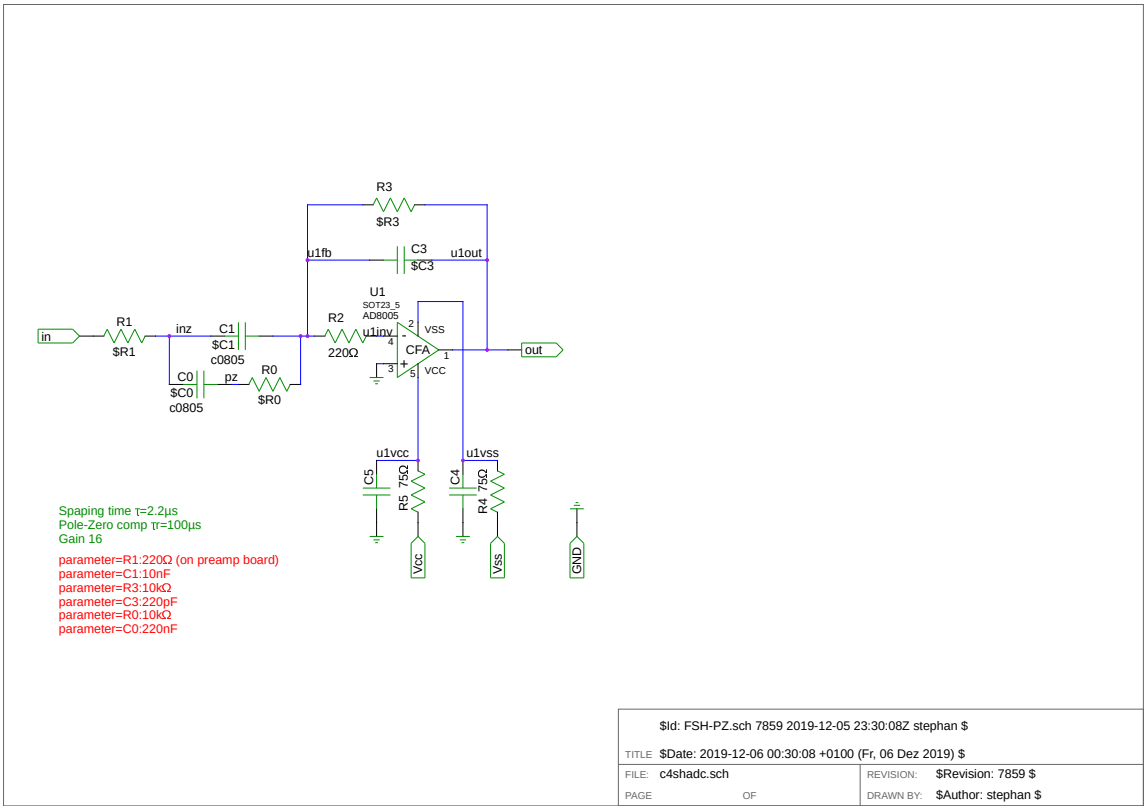

Figure 5: Shaper schematics. DORN parameter values are assigned on the toplevel Fig.3.

2.8.4 Sallen-Key Filter

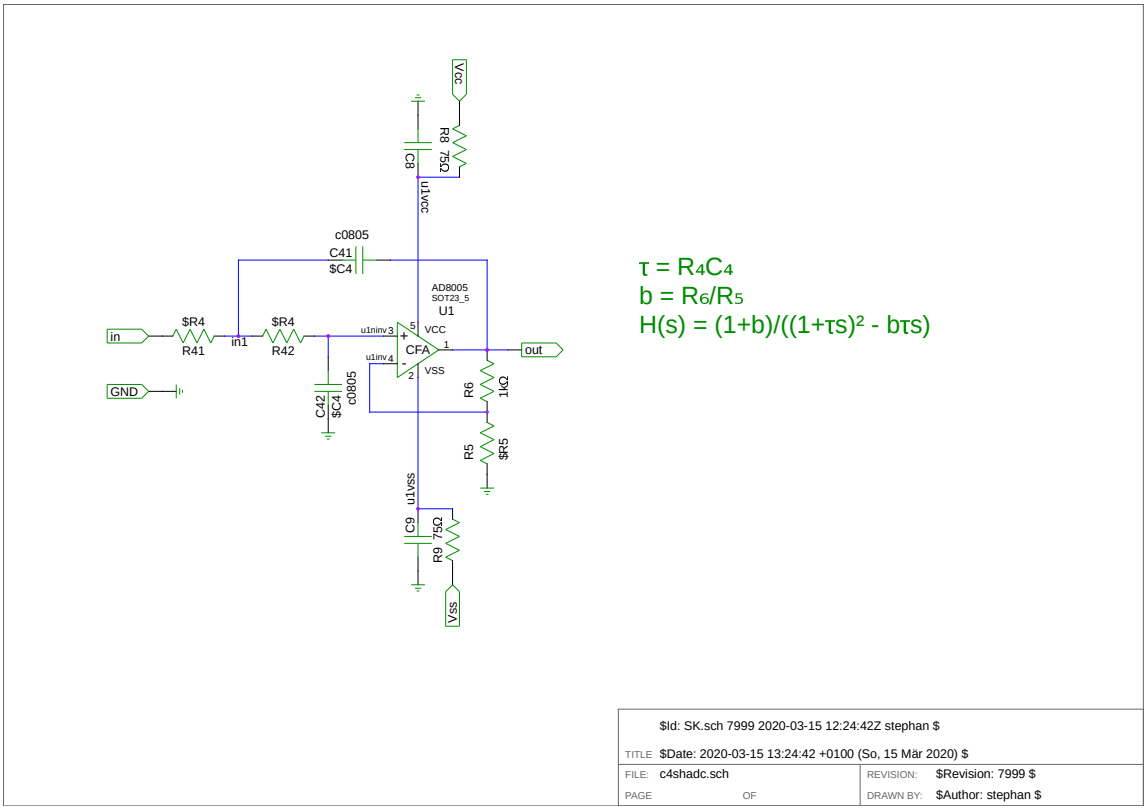

Figure 6: Sallen-Key Filter Schematics.

3 Digital Processing

The Verilog module `dorn_core` implements the digital processes to control the ADCs, capture the samples, detect signal above threshold, peak detection, apply a digital filter, and apply a phase correction to the pulse height. The implementation is a chain of three processes in modules `dorn_l1` to `dorn_l3`.

3.1 Level 1

When enabled, the ADCs are run continuosly, sampling with the maximum cadence of 1 μs supported by the ADC128S102. The multiplexer cycles through three input channels 0 to 2. Each channel is sampled with a cadence of 3 μs.

3.1.1 Sample Capture

The ADC output are deserialized and stored in a memory. There is space for eight samples in the memory, for each channel of each detector unit.

3.1.2 Trigger

Before a sample is written into memory it is compared to the value previously written to the same memory location. A trigger for the detector unit is issued when the new sample is more than a threshold above the sample obtained 24 μs earlier.

The current implementation uses the same set of three thresholds for each detector unit, one configurable threshold per channel.

3.2 Level 2

3.2.1 Derandomizing buffer

The second module provides a buffer for eight triggered events. Each event captures 16 samples from all channels of the triggered detector unit. The capture process can run in parallel for multiple detector units. Each trigger is assigned the next free event buffer, that is subsequently filled as the samples come in from the end of the level 1 pipeline. Start of capture is delayed by two samples from the time of trigger, so that only six samples before the trigger sample become available to the PHA filter.

Associated with each event buffer is stored the detector unit number, and a timestamp when the event entered the buffer.

3.2.2 Digital Filter

After a triggered event is fully captured in an event buffer it becomes available for filter processing. The filter operates on the oldest complete and unprocessed event. It computes two linear combinations over eight samples s_i, starting with the oldest eight samples ($i_0 = 0$) in the buffer

$$A = \sum_{i=0}^{7} a_i s_{i+i_0}, \tag{31}$$

$$B = \sum_{i=0}^{7} b_i s_{i+i_0}. \tag{32}$$

$$\tag{33}$$

The sum of the coefficients must be zero

$$\sum_{i=0}^{7} a_i = 0, \tag{34}$$

$$\sum_{i=0}^{7} b_i = 0. \tag{35}$$

$$\tag{36}$$

The filter continues to compute the same linear combinations over eight samples starting with the second oldest sample, $i_0 = 1$. The resulting value for A is compared to the previous value. This continues incrementing i_0 until a new A is smaller than the previous one, or the available samples are exhausted.

The current implementation provides a memory for individual coefficients for each channel for each detector. The filter employs one multiplier circuit.

The largest A found is provided to the level 3 module, together with the simultaneous result of B, a timestamp, detector unit and channel numbers. The timestamp is computed from the age of the event in the buffer, the sample position that yields the largest A, and a global clock.

3.3 Level 3

3.3.1 Phase

The next step is to compute the phase of the input pulse with respect to the sample cadence. The coefficients a_i and b_i shall be configured so that A gives a good estimate of the pulse height, and B is sensitive to the phase. The phase parameter ϕ is

$$\phi = \frac{B}{A}. \tag{37}$$

This division is computed serially in three steps. First, both inputs are normalized, taking the absolute value, and shifting the leading bit into the leftmost position. The division is performed by successive approximation with a multiplier circuit. Last, the result ist shifted, reversing the initial normalization and apply the sign.

3.3.2 Banana

A polynomial of third degree of ϕ is computed to obtain a correction factor b for the pulse height

$$\begin{aligned} \phi' &= \phi - p_0, && (38)\\ b(\phi') &= ((p_3\phi' + p_2)\phi' + p_1)\phi' + 1, && (39)\\ P &= bA. && (40) \end{aligned}$$

p_0 is the phase where A is expected to yield the largest value. The implementation uses one multiplier circuit. The result P is the final estimate of the pulse height. We need to achieve a precision significantly better than 1 %.

The coefficients p_i are stored in a memory individually for all detectors and channels.

The output of level 3 is ϕ, P and all its inputs. The user of the `dorn_core` is expected to capture this output into a FIFO and process it further with a μC.

3.4 Output

3.4.1 Raw Samples

The output of level 1 is also available to the user, to capture raw samples after a trigger, for diagnostics and calibration.

3.4.2 Housekeeping

The module can be instructed to interrupt normal processing, read all eight channels from all ADCs, and emit the data as housekeeping. We expect at least some of the extra ADC channels to be connected to supply voltages, detector bias current monitors, and NTC temperature sensors.

3.4.3 Lost Triggers

There is a signal `.lost()` that is active whenever a trigger cannot deliver data to an event buffer.

3.5 Configuration

The Verilog module is configurable at compile time via parameters and preprocessor symbols. At runtime configuration is controlled via registers and memories.

3.5.1 Compile Time Configuration

The module parameters are:

`DCLK=16` `dclk` frequency in MHz, multiple of 16.

`FO=1` Fanout, number of ADC128S102 output sets.

`ND=8` Number of detectors, 1 . . . DCLK/2.

`NC=3` Number of channels per detector, 1 . . . 8.

`NB=12` ADC resolution, 12 bits.

`NP1=8` Depth of the L1 pipeline, power of two.

`NP2=16` Depth of the L2 pipeline, power of two.

`NE=8` Depth of the L2 event buffer, power of two.

`NF=8` Length of the L2 filter, power of two.

Preprocessor symbols can be defined for

`WITH_FULL_L2_CONF` Use a memory for individual L2 filter coefficients. When undefined all channels use the same set of coefficients.

`WITH_FULL_L3_CONF` Use a memory for individual L3 banana coefficients. When undefined all channels use the same set of coefficients.

`WITH_L1_PEAK` When defined, an L1 trigger requires a maximum in the sequence of sampled ADC values. This restricts the filter coefficients to assume two samples after the maximum. Do not enable this option. Without this option, the trigger happens as soon as any sample exceeds the threshold.

3.5.2 Run Time Configuration

The communication with the `dorn_core` module is compatible with the IRENA configuration interface with 16-bit data buses. The module occupies an 11-bit address range. Runtime configuration items include level 1 trigger thresholds, trigger enable registers, Sample readout enable registers, level 2 coefficient memory, and level 3 coefficient memory. Individual registers are read/write, memories are write-only.

Detector trigger enable

1	p	0x808
-		*<trigger-enable>*

Multiple detectors trigger enable

1	p	0x809
-		*<multi-triggers>*

Samples readout trigger enable

1	p	0x80a
-		*<sa-triggers>*

Number of samples

1	p	0x80b
-		*<n-samples>*

Trigger thresholds

1	p	0x81	0 0	*<ch>*
-		*<threshold>*		

L2 filter coefficients

1	1	0 0 1 1 0	*<ch>*	*<det>*	*i*	*b*
-		a_i or b_i				

L3 banana coefficients

1	1	0 0 1 0 0 1 0	*<ch>*	*<det>*	*i*
p_i					

The *<threshold>* is in ADC channels. The banana polynom parameters are in units of $[p_0] = 2^{-14}$, $[p_1] = 2^{-16}$, $[p_2] = 2^{-14}$, and $[p_3] = 2^{-15}$.

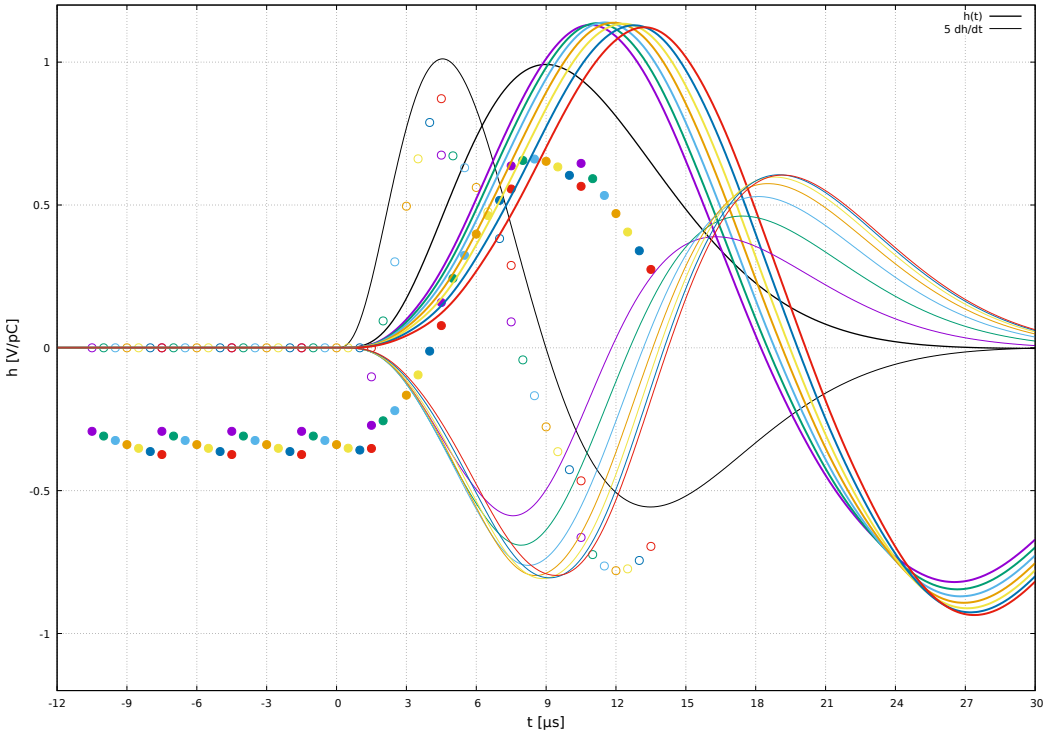

Figure 7: Seven sets of white noise filter coefficients and the resulting A (think lines, full circles) and B (thin lines, open circles). A and B are normalized to their respective white noise levels from Eq. (41)

Table 1: Filter Coefficients

	Set 2		Set 6	
i	a_i	b_i	a_i	b_i
0	-942	0	-1203	0
1	-942	0	-1203	0
2	-942	0	-1203	0
3	-942	0	-1186	0
4	-779	260	-39	2000
5	741	1857	1709	970
6	2000	-117	2000	-1082
7	1806	-2000	1125	-1888

4 Coefficients

Assuming a white noise spectrum where the noise of the ADC samples is uncorrelated, the natural choice of filter coefficients are a_i that follow the

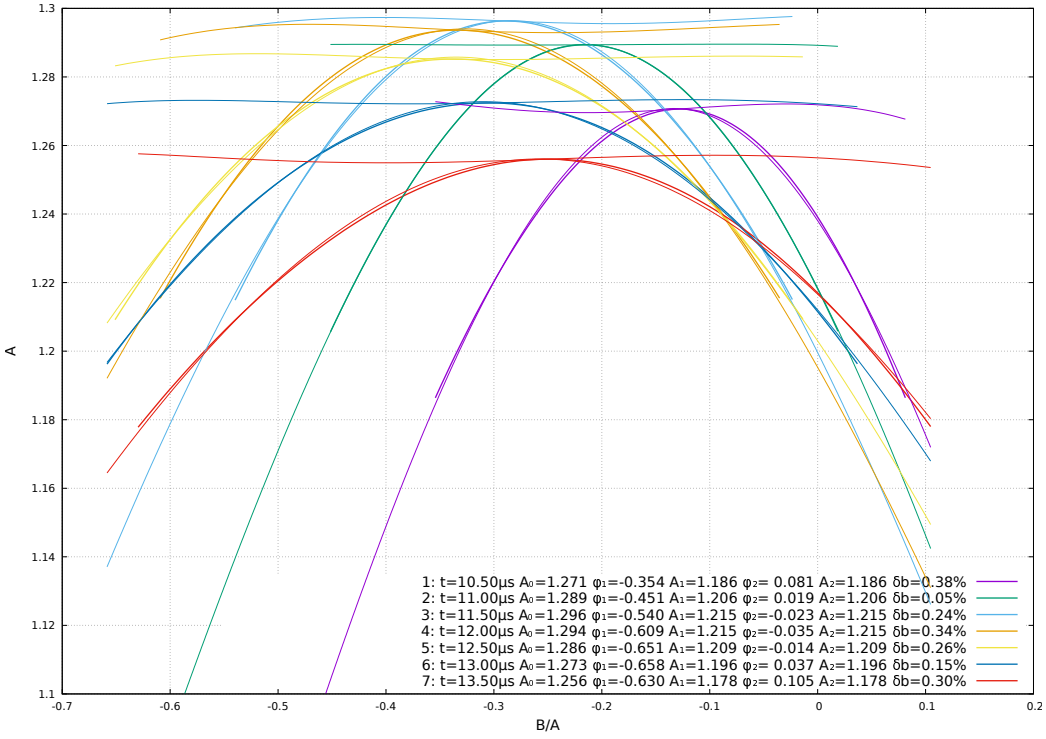

Figure 8: Banana Correction of the filter output from Fig. 7. Thick lines are the filter output. Thin lines are the polynomial fits and the corrected filter output. The numbers in the captions are t, reference time of the last coefficient, A_0, the peak height. (A_1, ϕ_1) and (A_2, ϕ_2) are the coordinates of the beginning and end of the banana curves. δ_b is the remaining pulse height variation after correction.

pulse shape, and b_i that follow the derivative of the pulse shape, with the mean value subtracted to satisfy Eq. (34). The first four coefficients b_i are set to zero. The noise level is the RMS of the coefficients.

$$n_a = \sqrt{\sum a_i^2} \tag{41}$$

$$n_b = \sqrt{\sum b_i^2} \tag{42}$$

Fig. 7 shows the results for seven sets of coefficients, where the reference time for the last coefficient varies from 10.5 μs to 13.5 μs. Fig 9 is the same plot zoomed in around the y-axis origin to illustrate the undershoot and secondary peak due to partial pole-zero compensation.

Fig. 8 shows the performance of the level 3 phase correction. Set 2 looks best in terms of remaining error δ_b but requires pretty old samples and op-

erates on a narrower range of B values than the runner up and favorite, set 6.

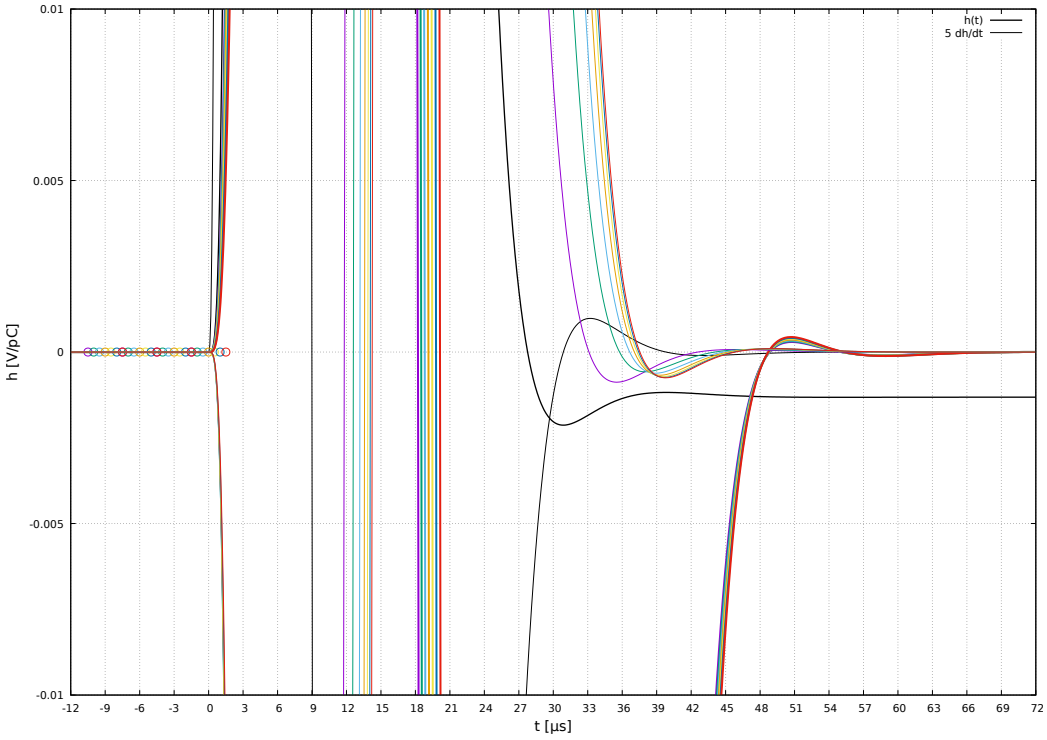

Figure 9: Undershoot of the pulse response and filter output. Zoom around the origin of the y-axis if Fig. 7.

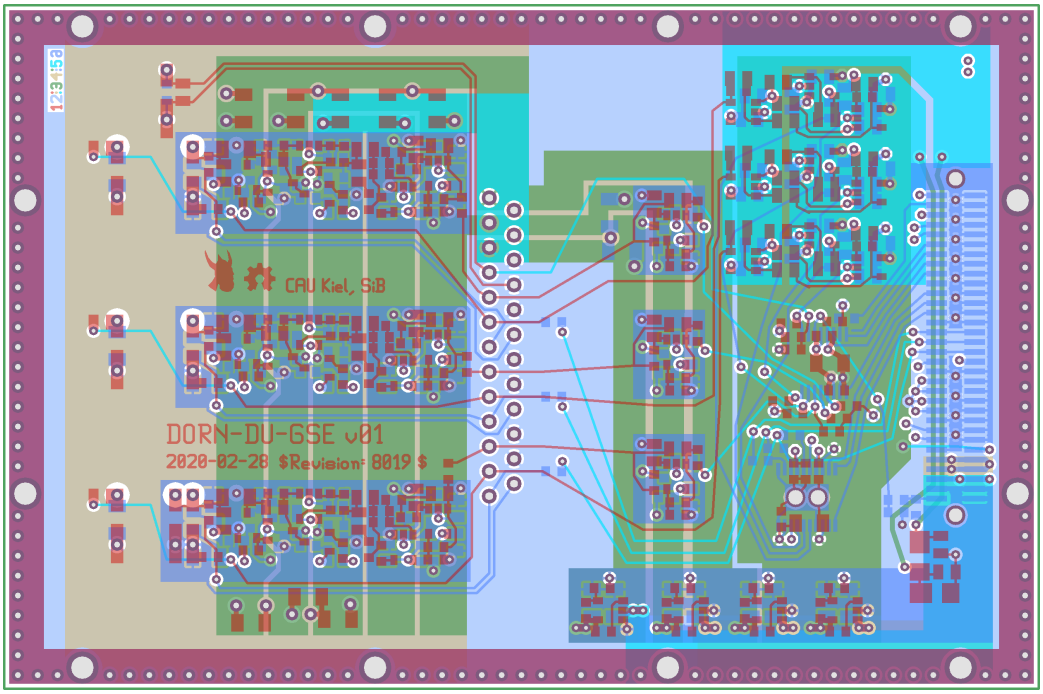

Figure 10: Layout of the `dorn-du-gse` board. The size is $106 \times 70\,\mathrm{mm}^2$

5 DARENA

A printed circuit board named `dorn-du-gse` was built with three instances of the analog chain, one ADC, three detector bias supplies, and three test pulsers. Bias supplies and pulsers are configured via an eight channel DAC. The board interfaces to an ARENA board with an Altera FPGA 3C25 and an Arm microcontroller with USB interface.

The `dorn-du-gse` board can be used partially populated either as GSE to test a detector unit, or as prototype electronics for a detector unit. For that purpose there are 25 connector pins available at the interface between the frontend and GSE parts.

5.1 Pulse Generator Test

The test pulse generator is driven by a pin of the FPGA switching from high-impedance to 2.5V. After $40\,\mu$s the pin returns to high-impedance. The pin is biased by a DAC via a resistor R_1. This signal is decoupled by a capacitor C_1 from the pulse injection input of the frontend circuit. The rise

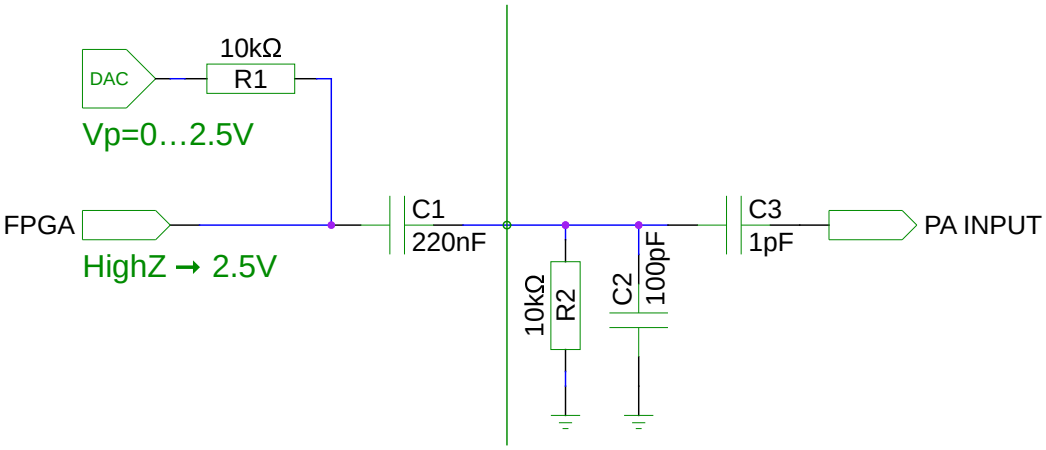

Figure 11: Schematics of the Test Pulse Generator.

time of the signal before the charge injection capacitor C_3 is about 300 ns. That is significantly slower than the assumed δ-function input in the analysis above.

The FPGA generates test pulses synchronous to the ADC convert clock. The phase is adjustable in 48 steps between 0 and 3 μs. A test was performed with a pulse amplitude of 20 %, with 1000 pulse delivered for each of the 48 phase setting, at 10 Hz repetition rate.

The pulses were delivered to channel 1 (center). A capacitor $C_{\text{det}} = ?$ pF was soldered in place of a detector. The FPGA board provides a built-in oscilloscope with 16 MHz sample rate and 14 bits resolution. The channel 1 input of the ADC was connected to the oscilloscope. The oscilloscope was configured to be triggered by the detector readout L1-trigger. The Oscilloscope adds a load of a few kΩ.

The test was performed on the bench without housing cover.

The data was captured in a file and analyzed.

5.1.1 Oscilloscope

Fig. 12 shows all oscilloscope samples. Since the L1-trigger, the pulse generator, and the oscilloscope all run from the same clock source, the signal is always sampled at the same amplitudes, as can be seen by the horizontal bands over all pulser phases. The time base is relative to the L1-trigger.

5.1.2 Banana Fit

Fig. 13 shows the outputs of the L2 and L3 processing. Light blue is the pulse height A, yellow is the phase channel B. From those the L3 processor

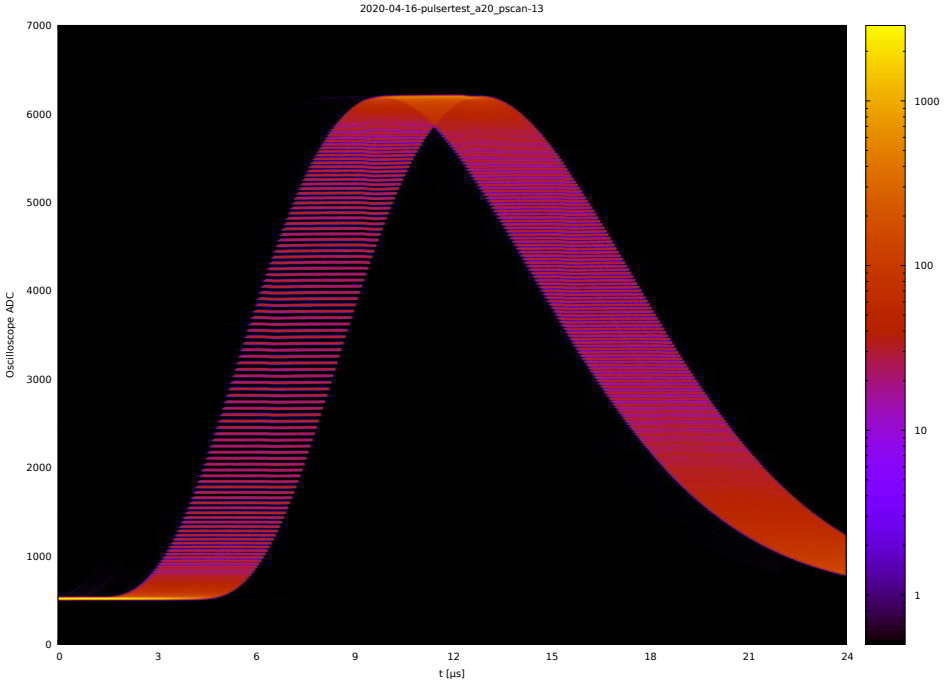

Figure 12: Oscilloscope view of the signal waveform at the input of the ADC.

derived the phase parameter B/A plotted in orange. Green is the polynomial to correct the amplitude, and purple is the corrected amplitude, as delivered by the L3 processor.

This result was obtained after several iterations of the test, after several bugs in the FPGA processing were fixed, a run with default polynomial parameters was used to fit proper banana parameters to the L2 output.

5.1.3 Resolution

Finally, the amplitudes from Fig. 13 were projected to the y-axis to fit the pulser resolution. Shown in Fig. 14 are the A amplitudes and the corrected amplitudes for each pulser phase and the total, and a Gauss fit to the corrected total. The obtained resolution is 0.1 %.

5.1.4 Samples

A further set of outputs are streams of ADC samples after each trigger. Those were plotted in Fig. 15. The time axis is derived from the sample timestamps and the pulser phase. The plot includes data from a test with twice the amplitude (40 %).

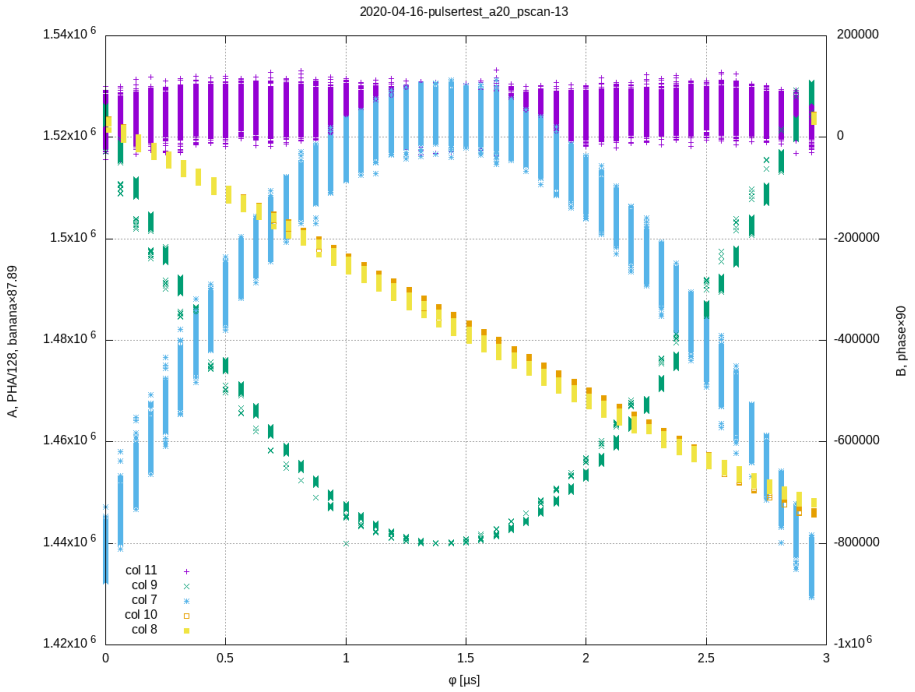

Figure 13: Banana plot.

You see that the analysis was done with a pulser setting at about one third of the full range, corresponding to about 7 MeV. The resolution is thus better than 10 keV.

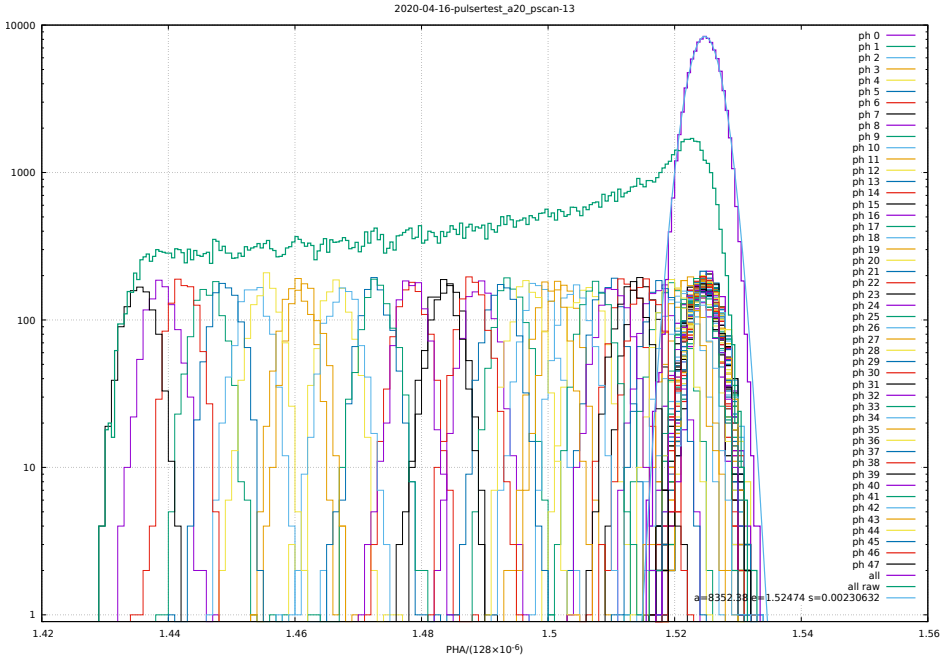

Figure 14: Pulser Spectrum.

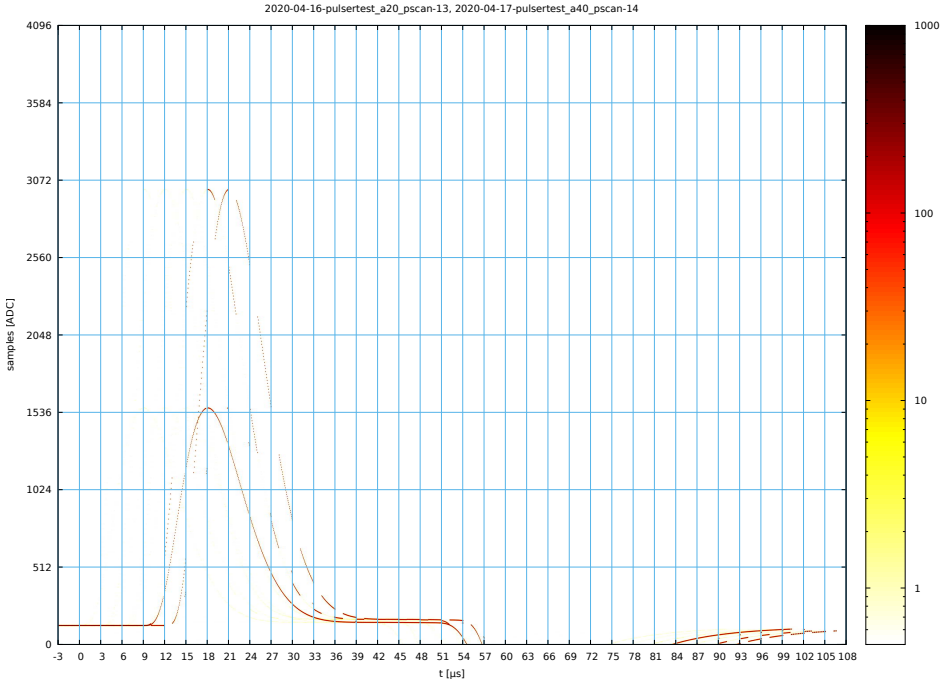

Figure 15: Samples.

6 Scripts

6.1 Transfer Function, maxima

```
display2d:true$

Hp: -1/Cp *taup/(1+s*taup);
hp: ilt(Hp,s,t);

HsZ: -Z2/Z1;
HsRC: HsZ, Z1=R1+1/(s*C1), Z2=R2/(1+s*C2*R2), ratsimp;
Hs: factor(ev(ev(HsRC, C1=tau/R1, C2=tau/R2), R2=a*R1, ratsimp));
hs: factor(ev(ilt(ev(Hp*Hs,taup=p*tau), s, t), ratsimp));

O1: U3 = Z3*I3;
O2: U4 = Z4*I4;
O3: U5 = Z5*I5;
O4: U6 = Z6*I6;
O5: U7 = Z7*I7;
O6: U8 = Z8*I8;
M1: Uin = U3+U4+U5;
M2: U5 = U7;
M3: U7+U8 = HfZZ*Uin;
M4: U4 = U6+U8;
K1: I3 = I4+I6;
K2: I4 = I5;
K3: I7 = I8;

GS: [O1,O2,O3,O4,O5,O6,M1,M2,M3,M4,K1,K2,K3];
UN: [I3,I4,I5,I6,I7,I8,U3,U4,U5,U6,U7,U8,HfZZ];

HfZ: factor(ev(rhs(solve(GS, UN)[1][13]), ratsimp));
Hf: factor(ev(ev(HfZ, Z3=Z4, Z5=Z6), Z6=Z4/s/tauf, Z8=b*Z7, ratsimp));

declare(b,real,b,real,p,real,tau,real,taup,real,taua,real,q,real);
assume(a>0, b>0, b<4, tau>0, p>0, q>0);

hf: factor(ev(ilt(ev(Hp*Hs*Hf,taup=p*tau,tauf=tau), s, t), ratsimp));

Ha: 1/(1+s*taua);
H: Hp*Hs*Hf*Ha, taup=p*tau, tauf=tau, taua=q*tau;
h: factor(ev(ilt(H, s, t), ratsimp));

Hd: 1/(1+s*d*tau)*H;
hd: factor(ev(ilt(Hd, s, t), ratsimp));

display2d:false$
hp;
hs;
```

```
hf;
h;
hd;
```

6.2 Banana Correction, gnuplot

```
# -*- gnuplot -*-

Cp = 1.0
tau = 2.7
taup = 100
p = 8000/tau
q = 1/tau
a = 2.7
b = 1/2.2
d = 1

load "dorn-shaper-h.gpt"

epsilon=0.01
dh(t)= 5 * (h_hk(t+epsilon)-h_hk(t-epsilon)) / (2*epsilon)

nF = 7
nC = 8
array aa[nF*nC]
array bb[nF*nC]

array nna[nF]
array nnb[nF]
array ass[nF]
array bss[nF]

t_range = 3.0
t_max = 13.5
t_step = t_range/(nF-1)
t_min = t_max - 7*3 - t_range

do for [n=1:nF] {
    aaa = 0
    bbb = 0
    do for [i=1:nC] {
        t = t_min + (n-1)*t_step + 3*(i-1)
        ii = nF*(i-1)+n
        aa[ii] = h(t)
        bb[ii] = i>4 ? dh(t) : 0
        aaa = aaa + aa[ii]
        bbb = bbb + bb[ii]
    }
```

```
    aaa = aaa/nC
    bbb = bbb/(nC-4)

    amax = 0
    bmax = 0
    do for [i=1:nC] {
        ii = nF*(i-1)+n
        aa[ii] = aa[ii] - aaa;
        if (i>4) {
           bb[ii] = bb[ii] - bbb;
        }
        if (abs(aa[ii])>amax) { amax=abs(aa[ii]) }
        if (abs(bb[ii])>bmax) { bmax=abs(bb[ii]) }
    }

    nna[n] = 0
    nnb[n] = 0
    aaa=0
    bbb=0
    ass[n]="a:"
    bss[n]="b: 0 0 0 0"
    amax = 2000/amax
    bmax = 2000/bmax
    do for [i=1:nC] {
        ii = nF*(i-1)+n
        aa[ii] = int(aa[ii]*amax)/amax
        aaa = aaa+aa[ii]
        if (i==nC) { aa[ii] = aa[ii]-aaa }
        ass[n] = ass[n].sprintf(" %.0f", aa[ii]*amax)
        if (i>4) {
           bb[ii] = int(bb[ii]*bmax)/bmax
           bbb = bbb+bb[ii]
           if (i==nC) { bb[ii] = bb[ii]-bbb }
           bss[n] = bss[n].sprintf(" %.0f", bb[ii]*bmax)
        }
        nna[n] = nna[n] + aa[ii]**2
        nnb[n] = nnb[n] + bb[ii]**2
    }
    print ass[n]
    print bss[n]
}

array tt[nC]
do for [i=1:nC] { tt[i] = t_min+3*(i-1) }

Aa(t, i) = h_hk(t)*aa[i] + (i<=nF ? 0 : Aa(t-3, i-nF))
Bb(t, i) = h_hk(t)*bb[i] + (i<=nF ? 0 : Bb(t-3, i-nF))

A(t, n) = Aa(t, nF*(nC-1)+n)
```

```
B(t, n) = Bb(t, nF*(nC-1)+n)
P(t, n) = B(t,n)/A(t,n)

if (1) {
reset
set xra[-12:30]
set xtics 3
set grid
set samples 1000
set xlabel "t [μs]" font ",16"
set yra [-1.2:1.2]
set ylabel "h [V/pC]"  font ",16"
plot h(x) tit "h(t)" w l lt 8 lw 2, \
     dh(x) tit "5 dh/dt" lt 8, \
     for [n=1:nF] tt u ($2+(n-1)*t_step):(aa[nF*($1-1)+n]) notit w p pt 7 lt n, \
     for [n=1:nF] tt u ($2+(n-1)*t_step):(bb[nF*($1-1)+n]) notit w p pt 6 lt n, \
     for [n=1:nF] A(x,n)/sqrt(nna[n]) notit w l lt n lw 3, \
     for [n=1:nF] B(x,n)/sqrt(nnb[n]) notit w l lt n

set term pdf size 11in,7.7in
set out "dorn-shaper-coeff_a4.pdf"
replot
unset out
set yra [-0.01:0.01]
set xra [-12:72]
set out "dorn-shaper-undershoot_a4.pdf"
replot
unset out
set term pop
set yra [-1.2:1.2]
set xra[-12:30]
replot
}

reset
array ttt[101]
do for [i=1:101] { ttt[i]=3*(i-51)/100. }

array oo[nF]
do for [n=1:nF] {
   oo[n] = t_min + 19.5 + n*t_step
   while (A(ttt[1]+oo[n],n) > A(ttt[101]+oo[n],n)) { oo[n] = oo[n]-0.1 }
   while (A(ttt[1]+oo[n],n) < A(ttt[101]+oo[n],n)) { oo[n] = oo[n]+0.01 }
   while (A(ttt[1]+oo[n],n) > A(ttt[101]+oo[n],n)) { oo[n] = oo[n]-0.001 }
}

print oo
#pause -1 "hit <return>"
```

```
if (1) {
array a0[nF]
array p0[nF]
array p1[nF]
array p2[nF]
array p3[nF]
banana(phi,n) = ((p3[n]*(phi-p0[n]) + p2[n])*(phi-p0[n]) + p1[n])*(phi-p0[n]) + 1
banana_(phi) = ((p3_*(phi-p0_) + p2_)*(phi-p0_) + p1_)*(phi-p0_) + 1
do for [n=1:nF] {
    p0_ = B(oo[n],n)/A(oo[n],n)
    p1_ = 0.01
    p2_ = 1.0
    p3_ = 0.001
    a0_ = A(oo[n],n)
    fit a0_/banana_(x) ttt u (P($2+oo[n],n)):(A($2+oo[n],n)) via p2_
    fit a0_/banana_(x) ttt u (P($2+oo[n],n)):(A($2+oo[n],n)) via p2_,p1_,p0_,a0_
    fit a0_/banana_(x) ttt u (P($2+oo[n],n)):(A($2+oo[n],n)) via p3_,p2_,p1_,p0_,a0_
    p0[n] = p0_
    p1[n] = p1_
    p2[n] = p2_
    p3[n] = p3_
    a0[n] = a0_
}
}
array bkey[nF]
do for [n=1:nF] {
    db1=1
    db2=-1
    do for [i=1:101] {
        t = ttt[i]+oo[n]
        db = a0[n]/banana(B(t,n)/A(t,n),n) - A(t,n)
        if (db<db1) { db1=db }
        if (db>db2) { db2=db }
    }
    bkey[n] = sprintf("%d: t=%5.2fμs A0=%.3f φ1=%6.3f A1=%.3f φ2=%6.3f A2=%.3f δb=%.2f%%", \
    n, t_min + (n-1)*t_step + 3*7, a0[n], \
    B(ttt[1]+oo[n],n)/A(ttt[1]+oo[n],n), (A(ttt[1]+oo[n],n)), \
    B(ttt[101]+oo[n],n)/A(ttt[101]+oo[n],n), (A(ttt[101]+oo[n],n)), \
    100*(db2-db1)/a0[n] )
    print bkey[n]
}
do for [n=1:nF] {
    print n, ass[n], "   ", bss[n]
}
set xra[*:*]
set grid
set samples 1000
set xlabel "B/A" font ",16"
```

```
set yra [1.1:1.3]
set ylabel "A" font ",16"
set key bottom right font ",16"

plot \
 for [n=1:nF] ttt u (P($2+oo[n],n)):(A($2+oo[n],n)) \
              tit bkey[n] w l lt n lw 2, \
 for [n=1:nF] a0[n]/banana(x,n) not w l lt n, \
 for [n=1:nF] ttt u (P($2+oo[n],n)):(A($2+oo[n],n))*banana(P($2+oo[n],n),n) \
              not w l lt n

if (1) {
set term pdf size 11in,7.7in
set out "dorn-shaper-banana_a4.pdf"
replot
unset out
set term pop
}
```