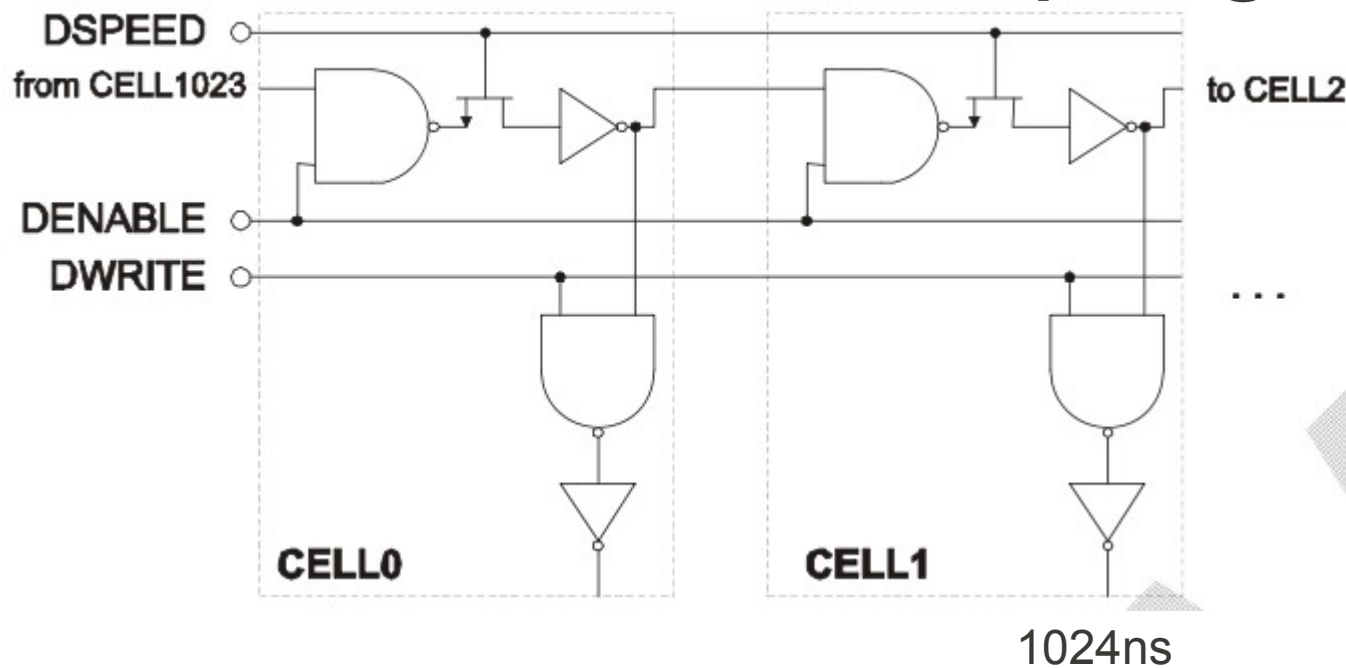


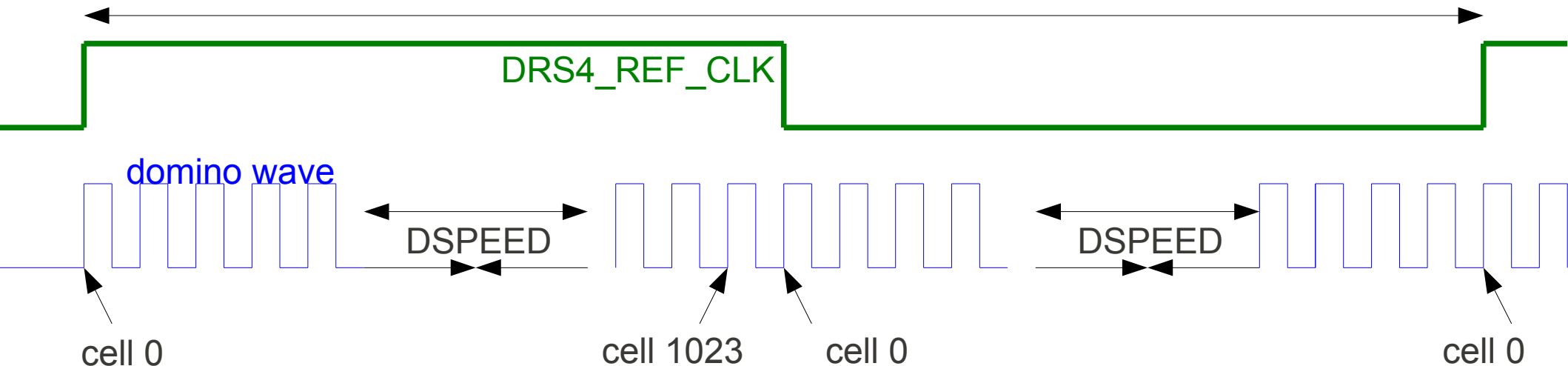
Crosscheck: Calculation of DRS4 time calibration constants

- What do we have
- How to overcome the problem
- What we currently do
- How well it works

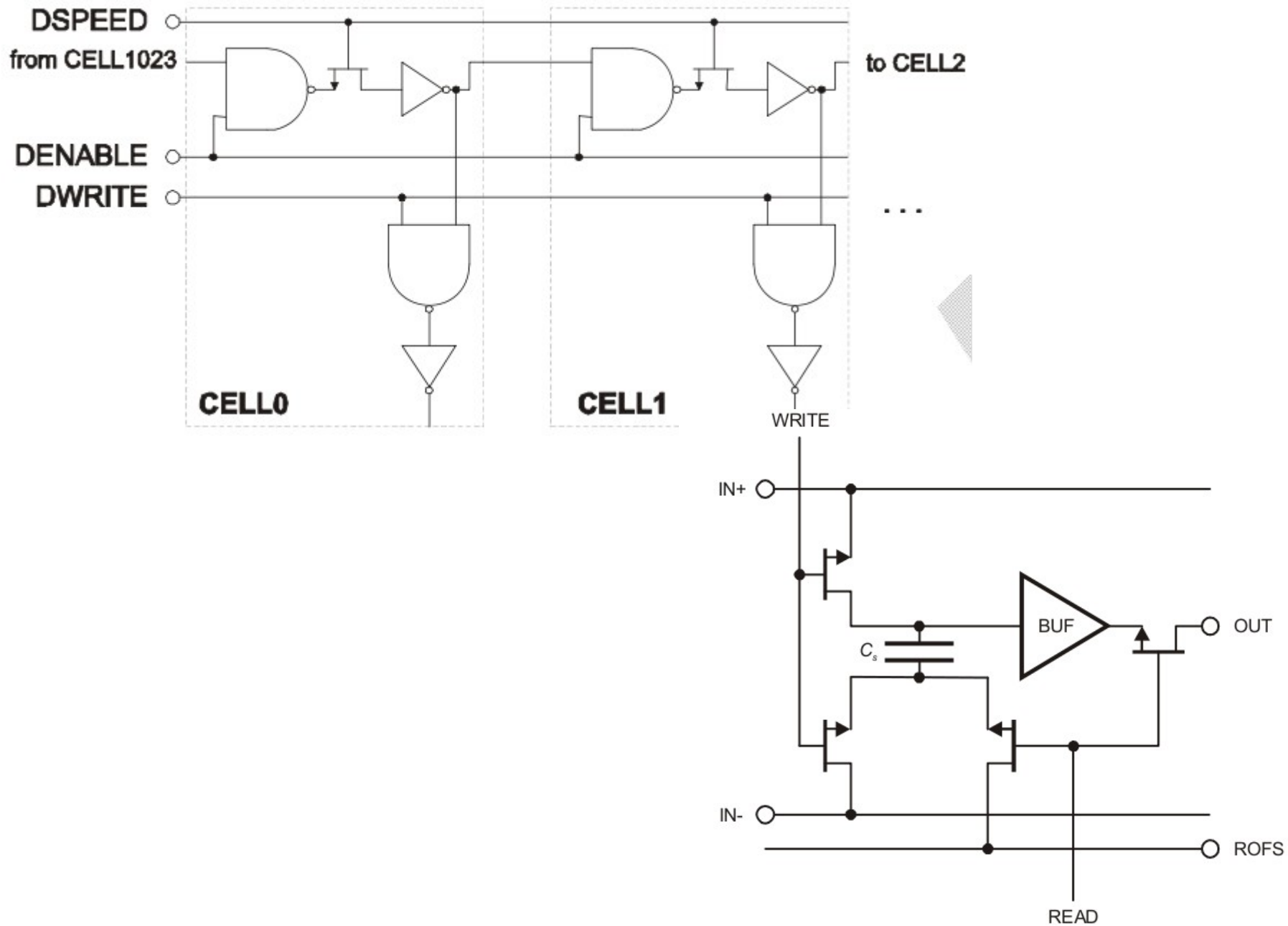
2GHz sampling how?



$$f_s = 2048 \cdot f_{ref}$$
$$f_{ref} \approx 976.56\text{kHz}$$



readout cell



Problem:

Probably because of inhomogeneities in the doping of the NMOS transistors between each NAND and NOT in the domino chain the time to switch from one cell to another is not homogeneously distributed within one DRS4 channel.

In order to achieve a close to perfect timing resolution we want to calibrate for this effect.

But how **exactly** should we do it?

What can be done?

inside the DRS4, only one channel for each DRS4 chip needs to be calibrated. One possibility to do this is to sample a high accurate sine wave with the DRS4 chip, and look for deviations between the sampled waveform and the ideal one obtained from a sine fit of all samples. Averaging over many waveforms at different phases of the sine wave, the fixed pattern aperture jitter can be measured and stored for calibration in a database for example.

An additional complication might arise from the fact that

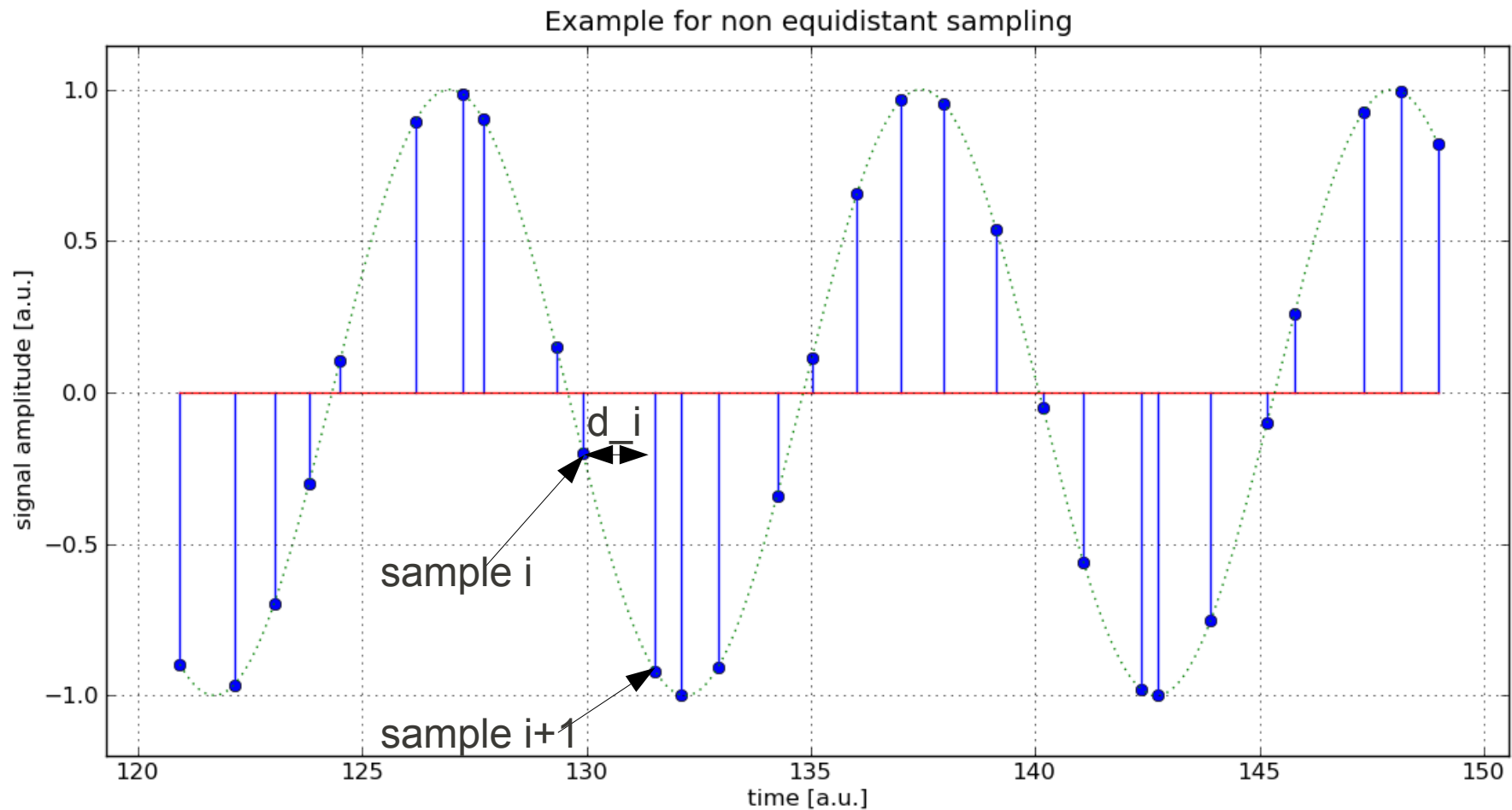
[DRS4 datasheet rev 0.9]

- In FACT we do not have an incredibly precise sine-wave at our disposal.
- We have a 250MHz (cable filtered) square wave, i.e. something between square and sine wave ;-)
- The phases of calibration signal and domino wave are **fixed**. (not good)

So what are we currently doing?

- We take 'drs-time' runs of 1000 events.
 - fixed phase: calibration signal \leftrightarrow domino wave
 - variable DRS start cell.
 - 250MHz, i.e. 8 samples = 1 calibration period
- We find the zero crossings of the calibration signal and thus e.g. the 'apparent period'.
- By averaging over many events we want to measure the relative 'size' of certain intervals of a DRS4 channel.

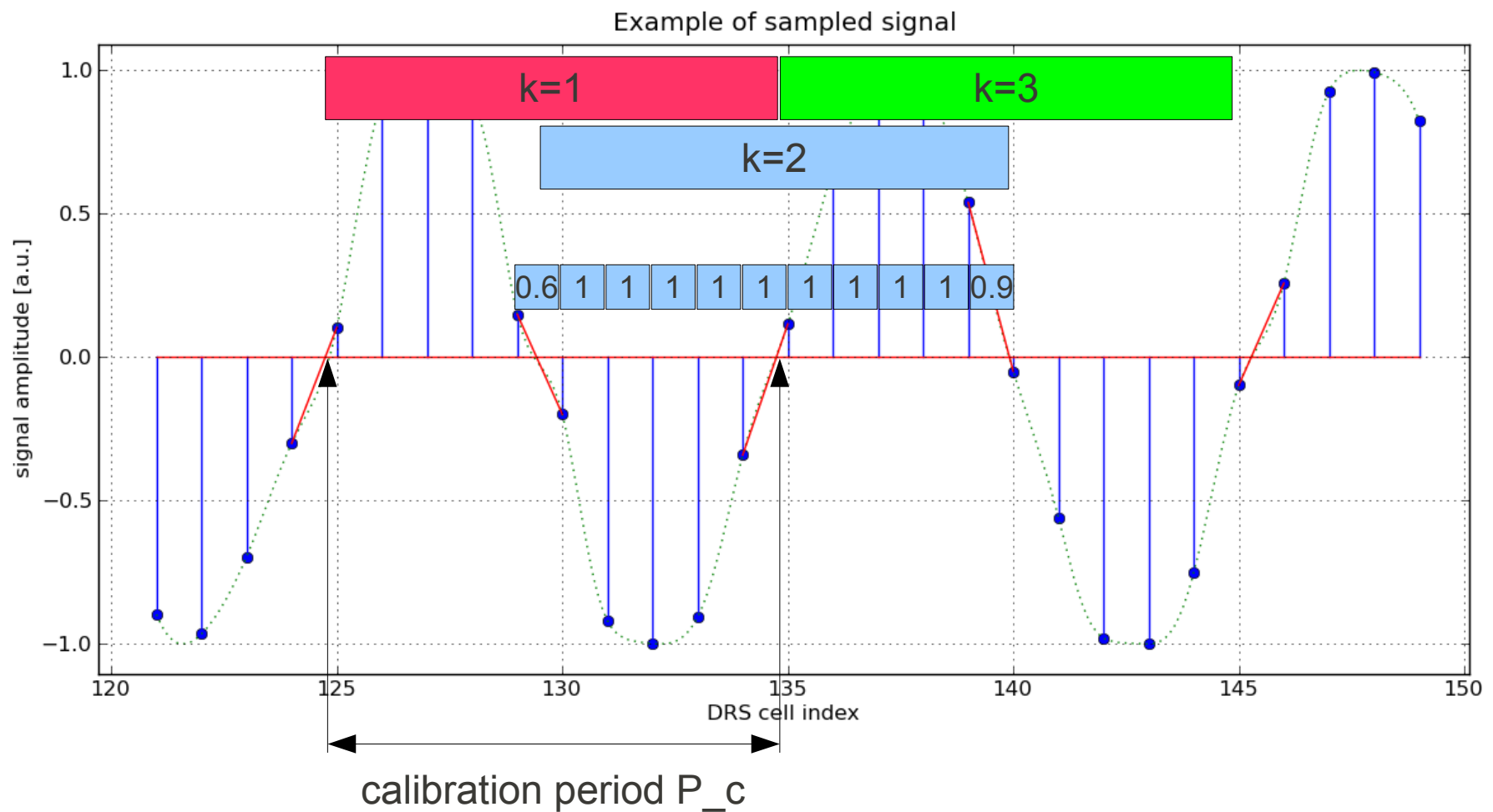
Let's define a common language \rightarrow



Let's call the temporal distance between consecutive samples:
sampling delay $\mathbf{d_i}$.
So sample $i+1$ will be taken $\mathbf{d_i[ns]}$ after sample i .

We already know the mean of all $\mathbf{d_i}$. Since the sum is fixed to the half period of the reference clk:

$$\sum_{i=0}^{1023} d_i = P_{RefClk}/2 \rightarrow \overline{d_i} = 0.5ns$$



We find the roots of the calibration signal by simple linear interpolation. A weight w is assigned to each delay d , according to its influence on the measurement of the period in question.

Each period we see, forms an independent measurement k .

I.e. : From each measurement k we learn:

$$\sum_{i=0}^{1023} d_i \cdot w_{ik} = P_c$$

Example $k=2 \rightarrow w[129:139] = (0.6, 1, 1, \dots, 1, 0.9)$

What is done in DrsCalibrateTime?

- Find the weights w_{ik}

- Define:
$$l_k = \sum_{i=0}^{1023} w_{ik} \quad w_i = \sum_{k=0}^{N_k} w_{ik} \quad wl_i = \sum_{k=0}^{N_k} w_{ik} \cdot l_k$$

- And:
$$s_n = \left(\sum_{i=0}^{n-1} wl_i \right) / \left(\sum_{i=0}^{n-1} w_i \right)$$

- With: $s = s_{1024}$

Original Code Comment:

```
// First calculate the average length s of a single  
// zero-crossing interval in the whole range [0;1023]  
// (which is identical to the/ wavelength of the  
// calibration signal)
```

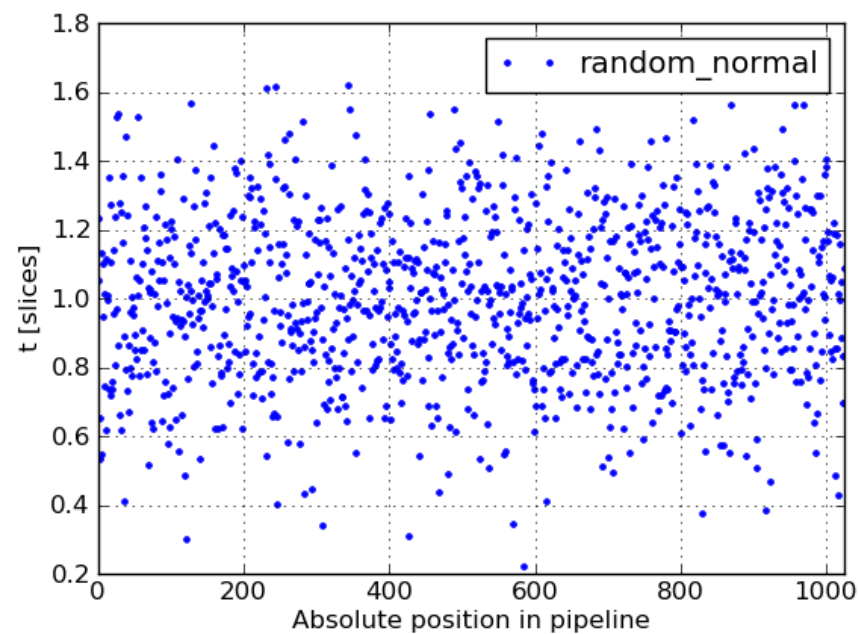
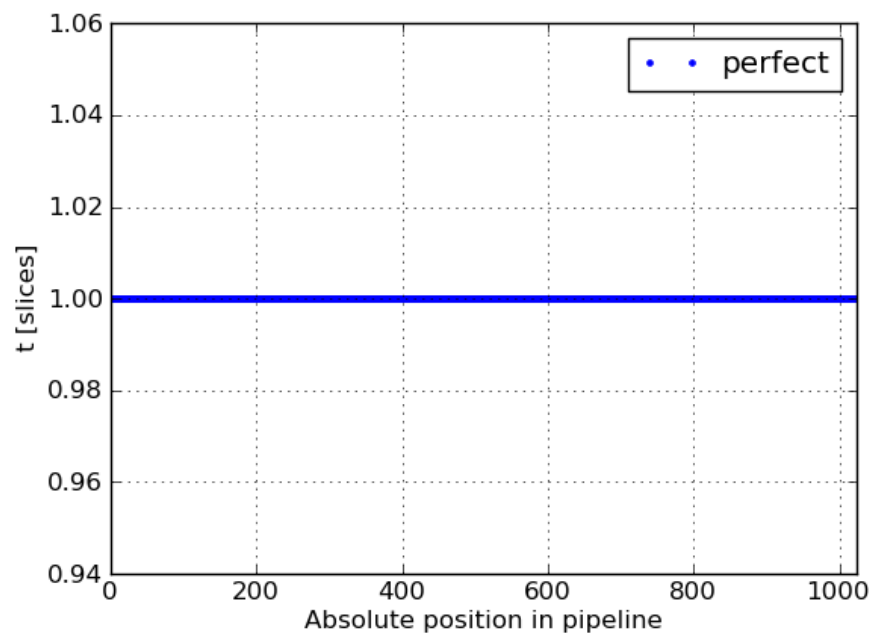
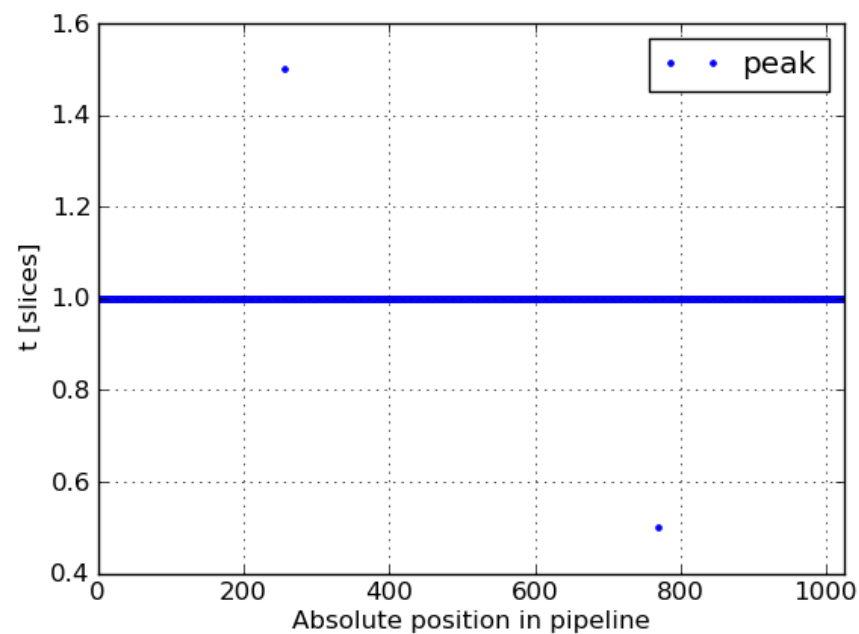
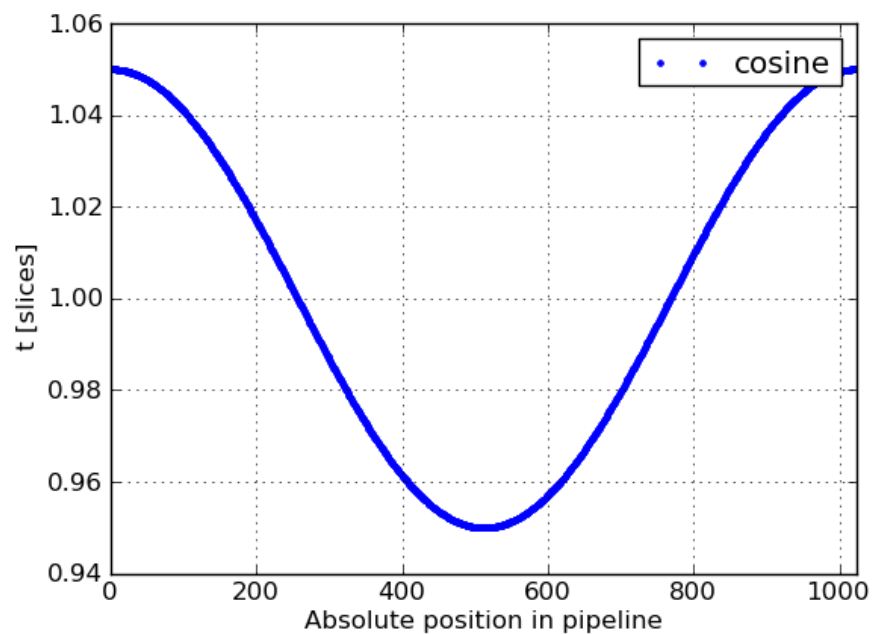
```
// Dividing the average length s of the zero-crossing  
// interval in the range [0;1023] by the average length  
// in the interval [0;n] yields the relative size of  
// the interval in the range [0;n].  
// The offset (defined as 'ideal - real') is then calculated  
// as  $n \cdot (1 - s/s_n) = o_n$ 
```

$$o_n = n \cdot (1 - s/s_n)$$

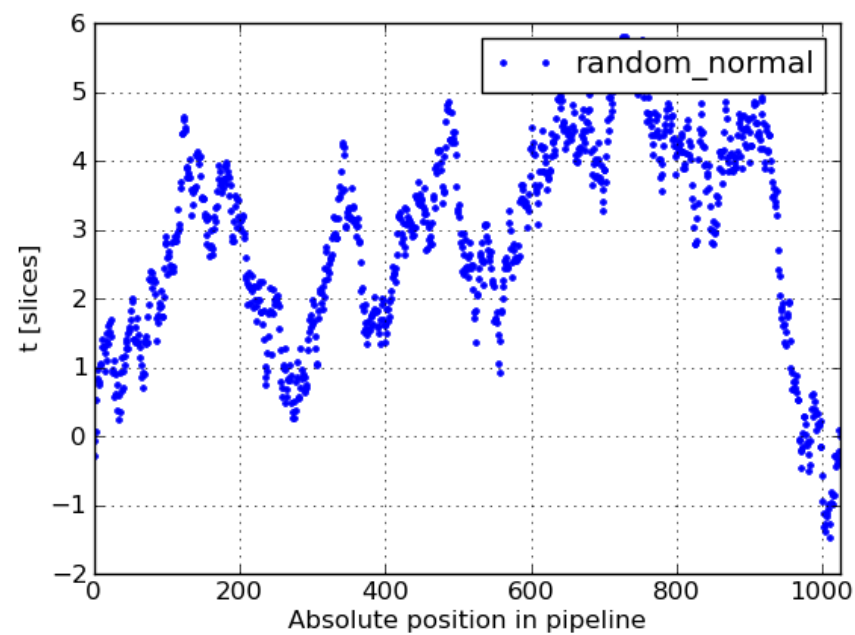
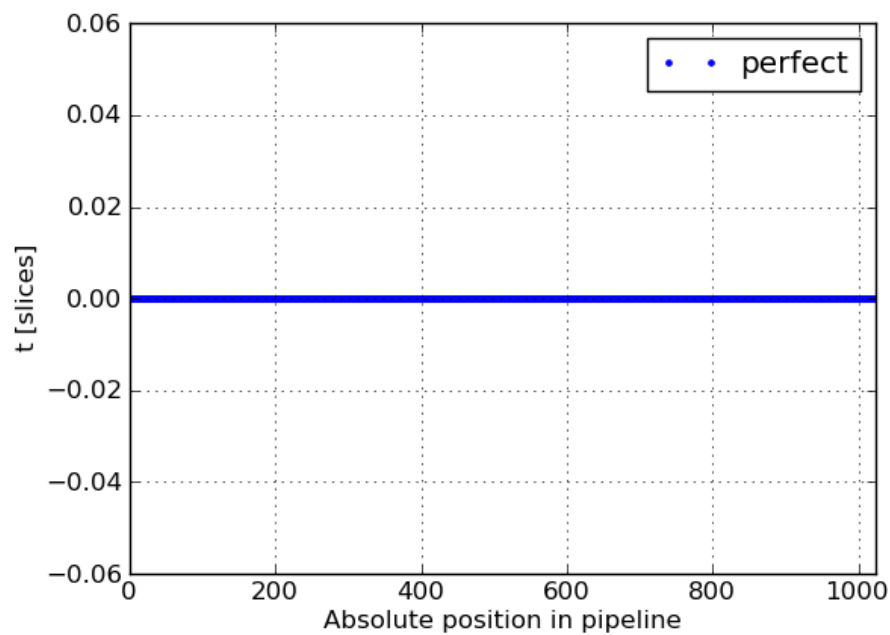
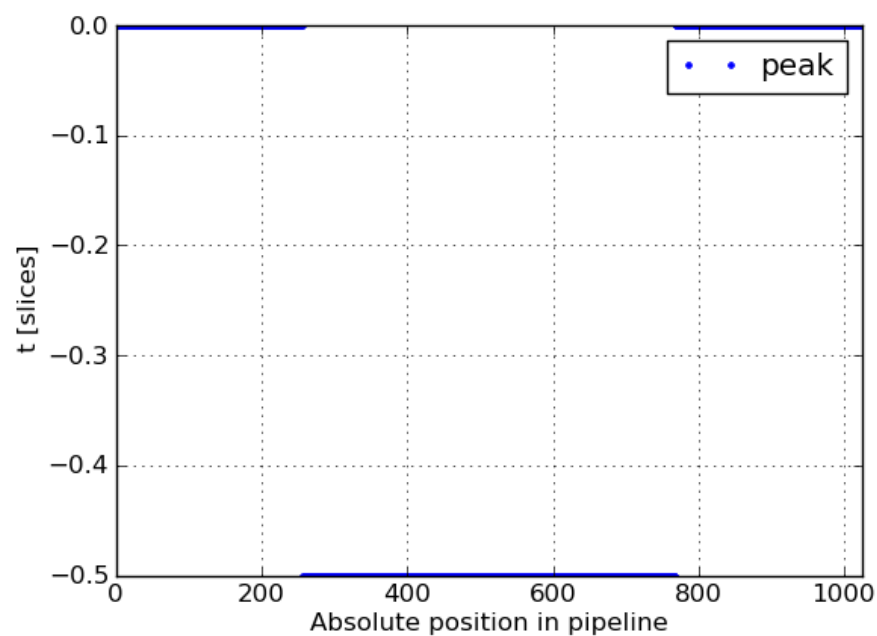
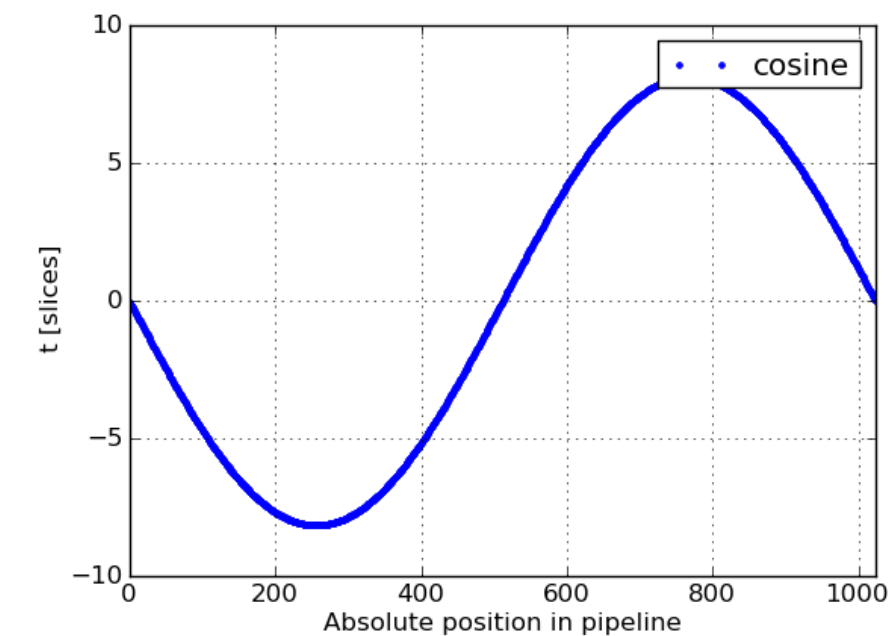
Testing!

- Choose some sampling delay distributions:
e.g. (flat, cosine, random, with_peaks)
- Choose a calibration signal (sine-wave) with:
 $f=250\text{MHz}$, $A=100\text{mV}$, $\text{phase}=\text{const.}$
- Sample it accordingly and add a bit of noise
($\text{rms}=2\text{mV}$)
- Write the result into a FITS file, so we can feed it to a stripped down callisto-version.

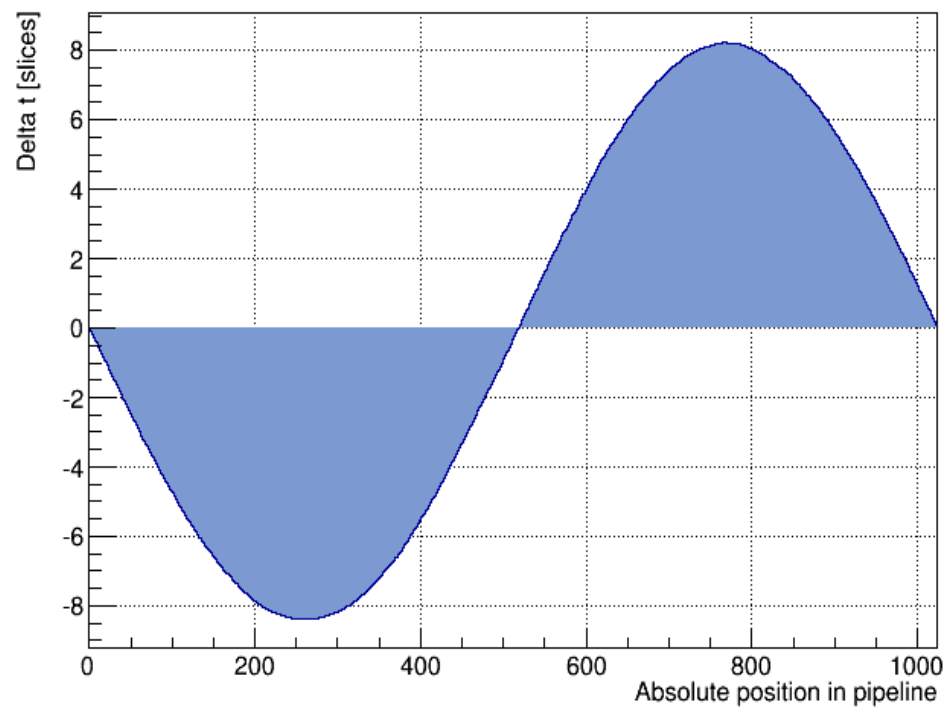
DNL : sampling delays



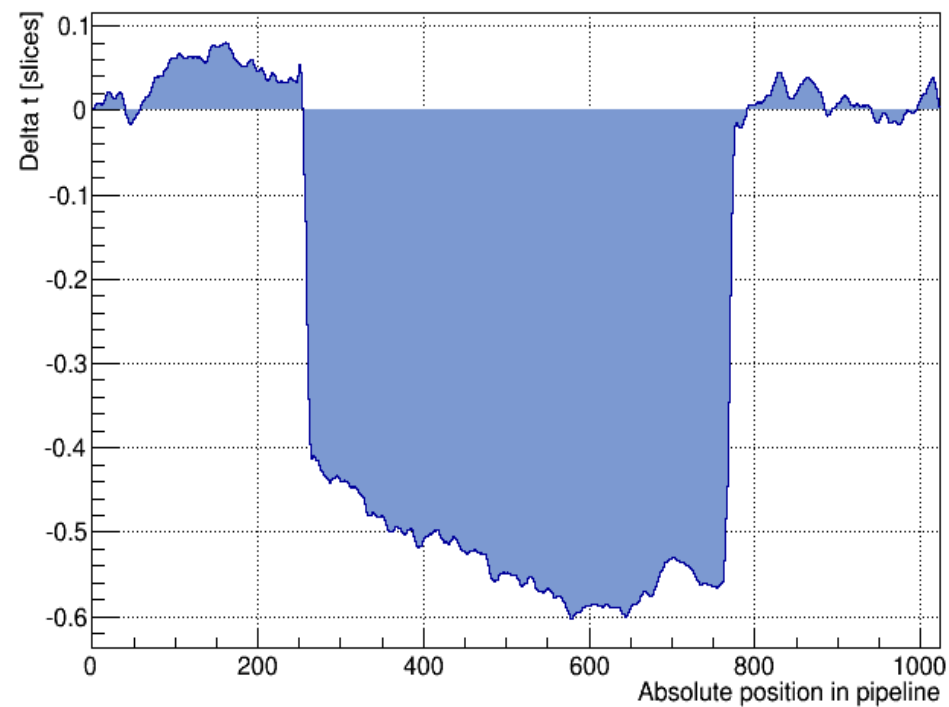
INL : (nominal-actual) sampling times



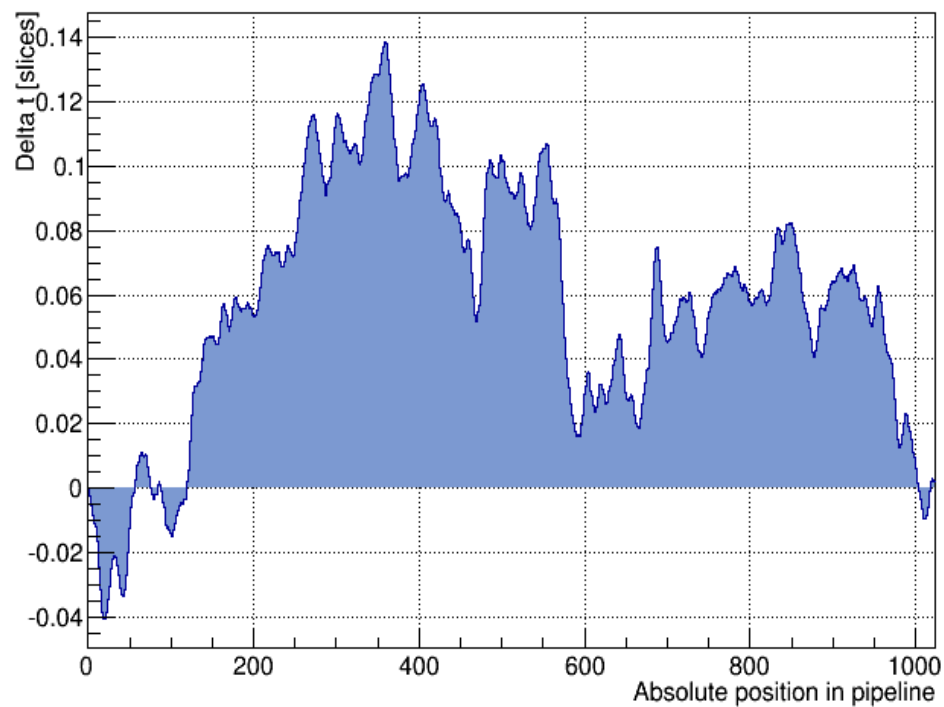
INL : cosine



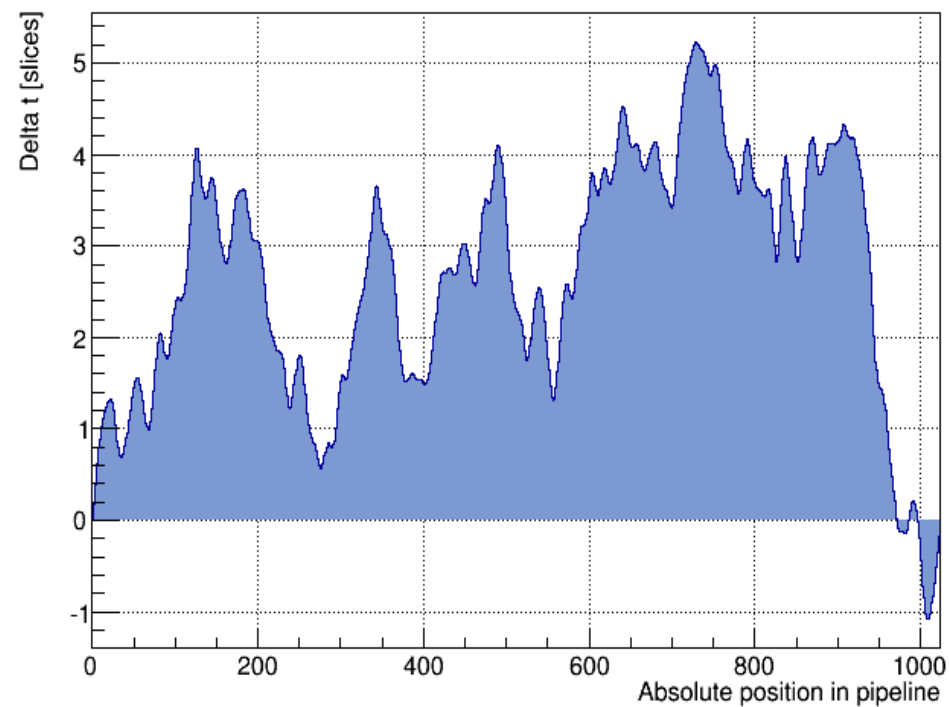
INL : peak



INL : perfect



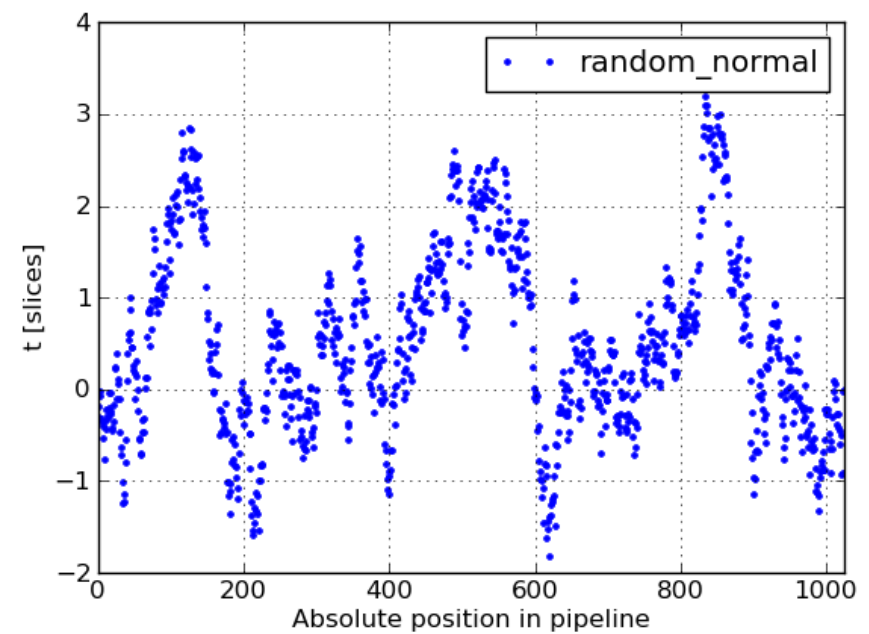
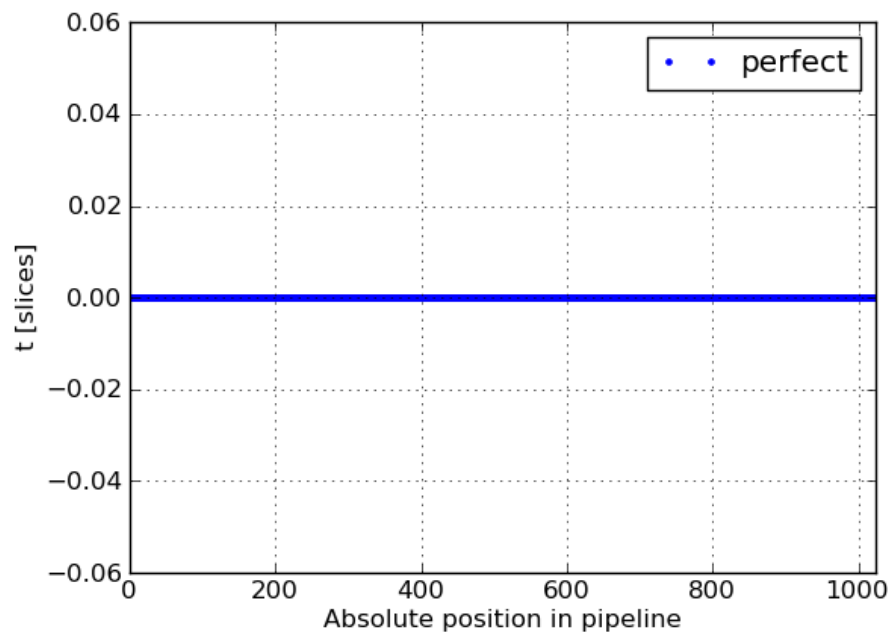
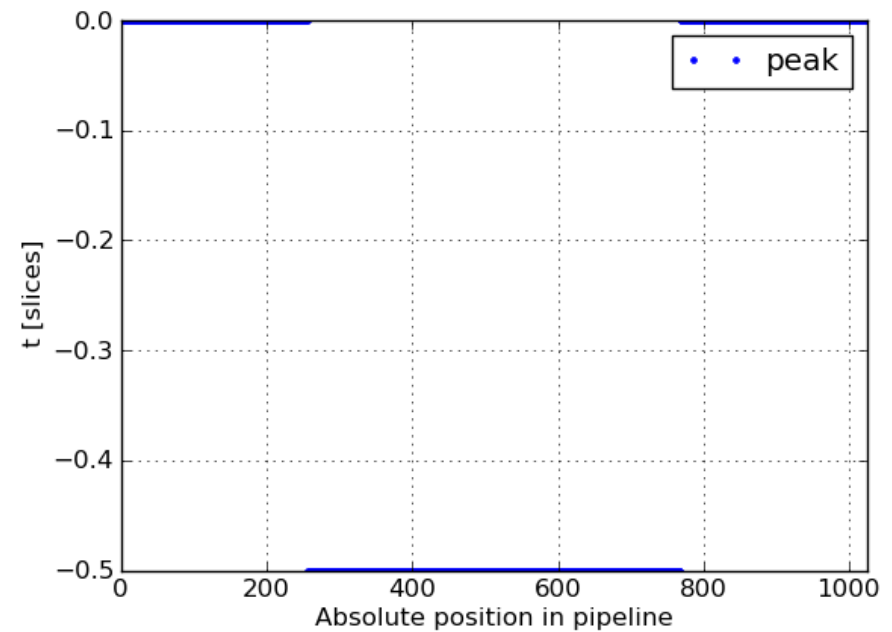
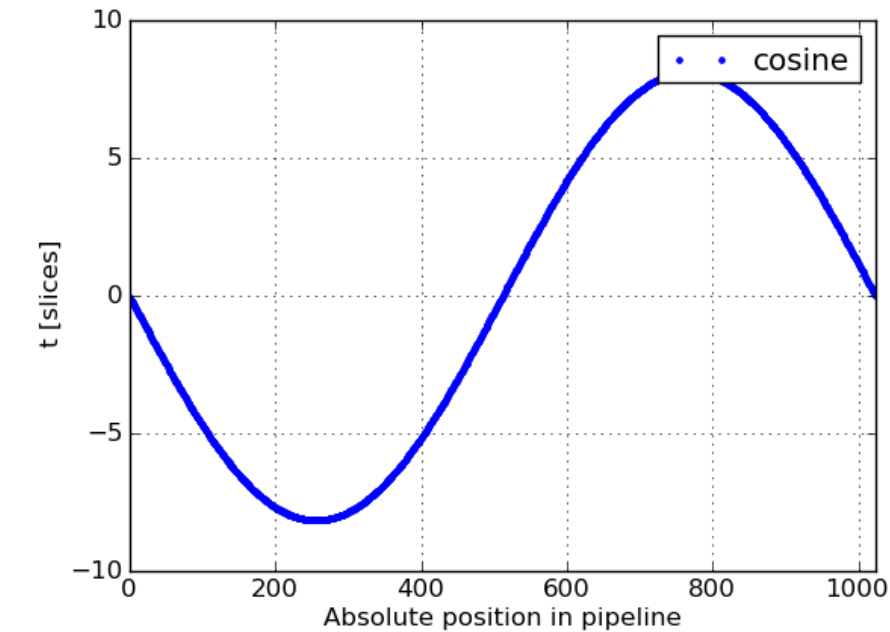
INL : random_normal



Maybe it would be better, if
we had a varying phase.

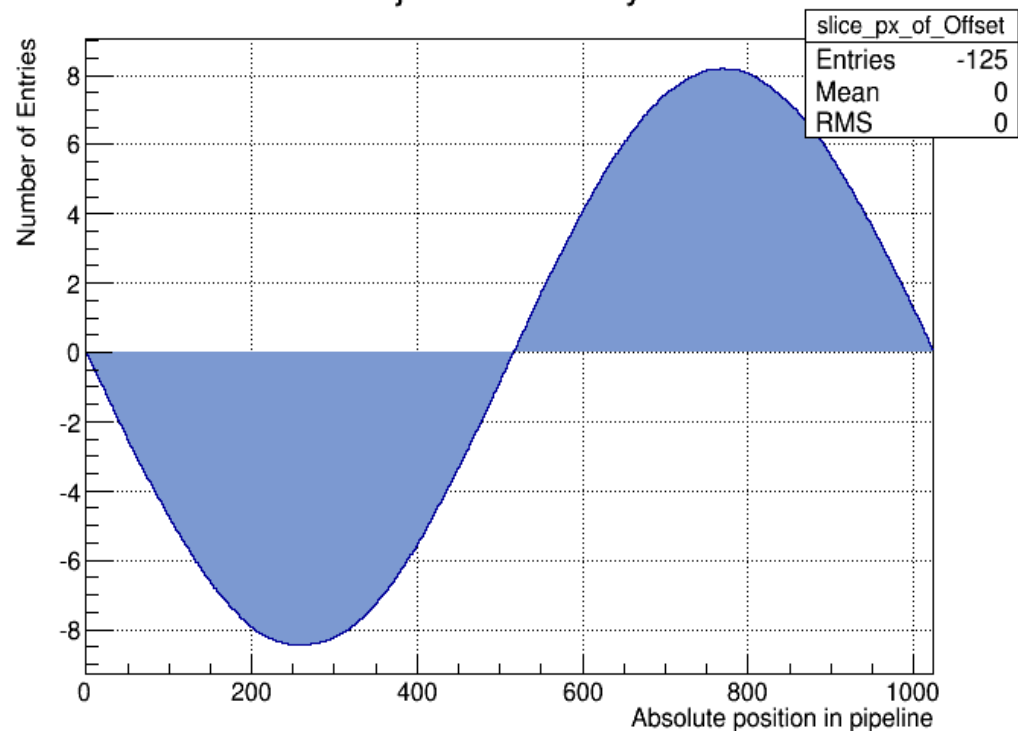
So I tried it ... again 1000 events:
but with varying phase between calibration
signal and domino wave.

INL : (nominal-actual) sampling times

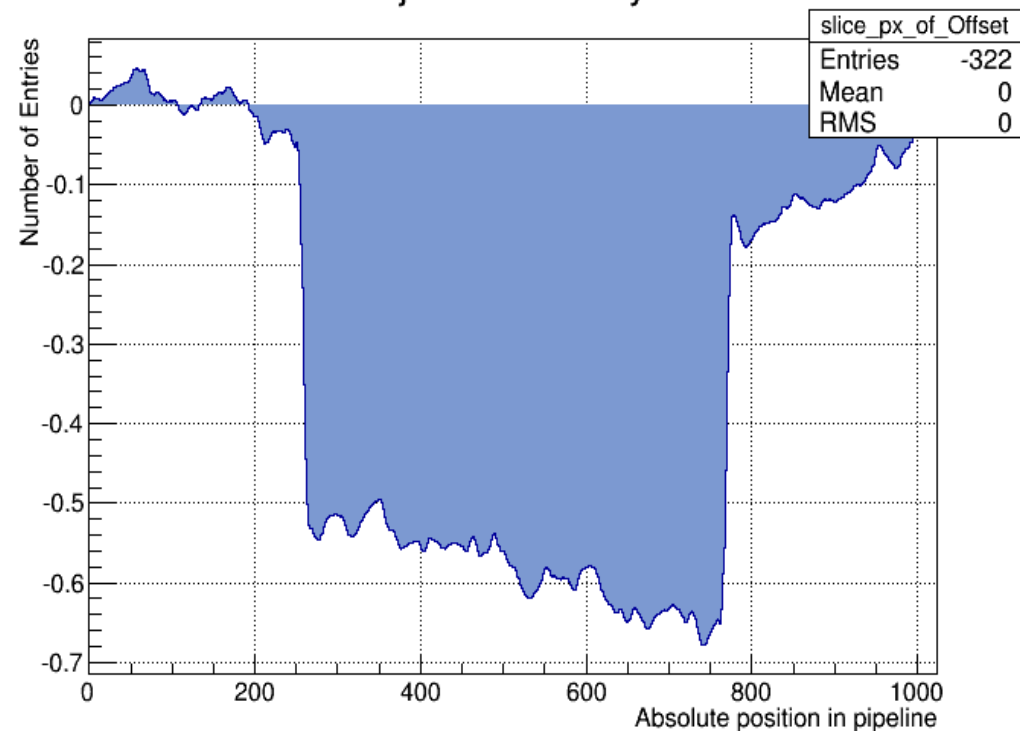


random_normal changed
due to non fixed seed!

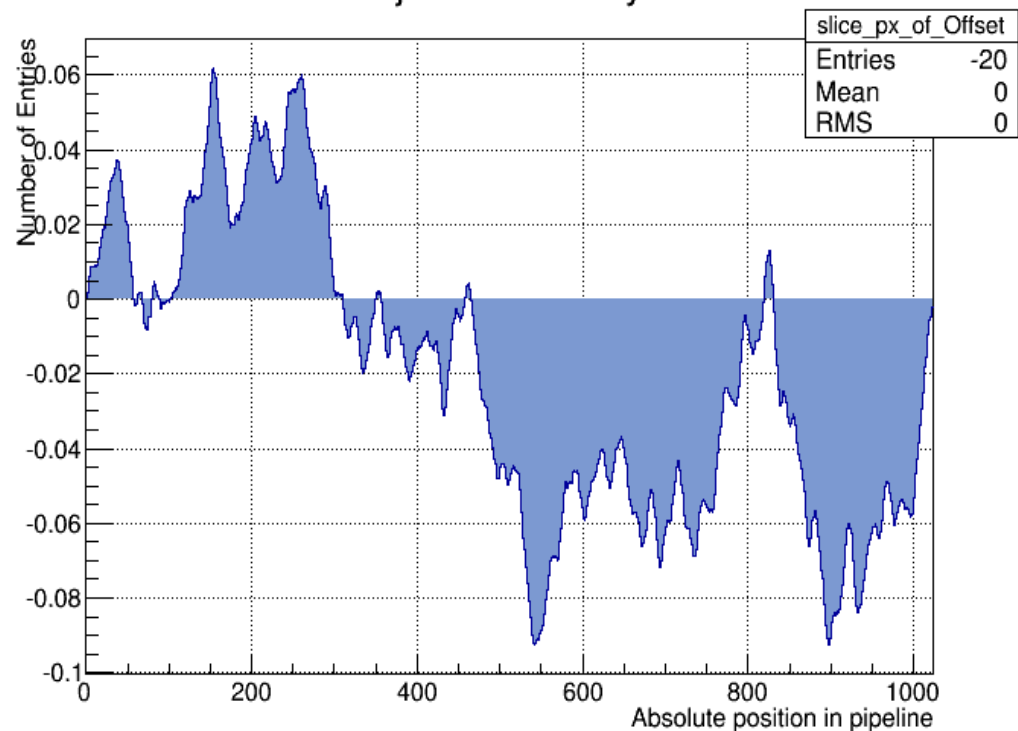
ProjectionX of biny=1



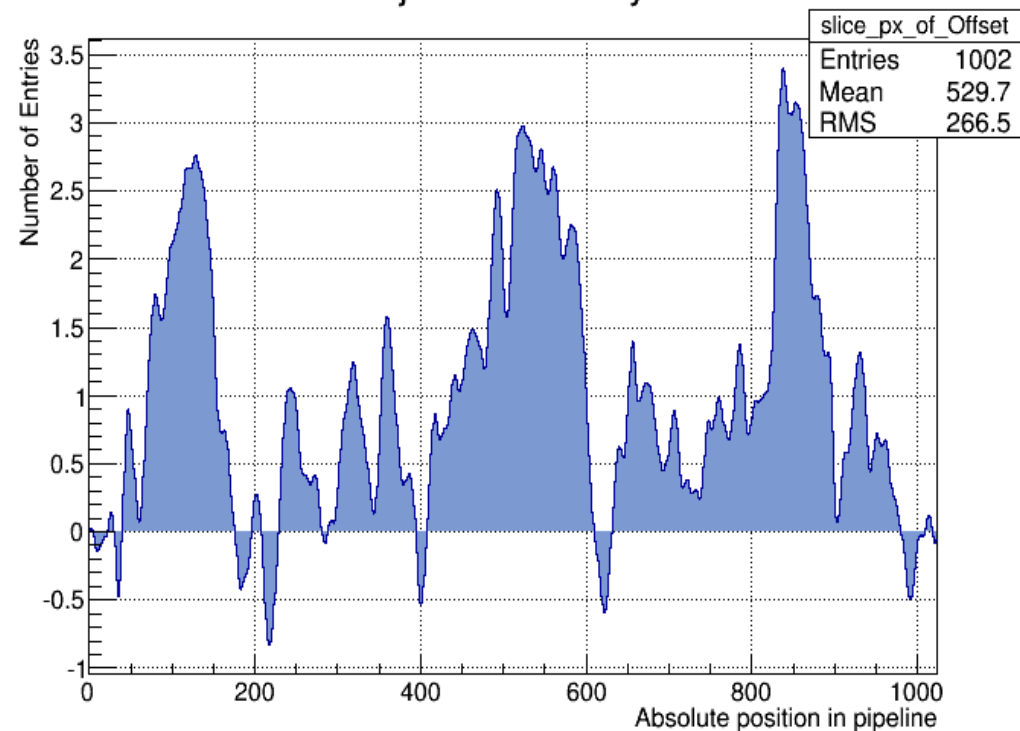
ProjectionX of biny=1



ProjectionX of biny=1



ProjectionX of biny=1



result:

- Both the variable phase as well as the constant phase input create reasonable results.
- We have no error estimation yet, since the magnitude of the deviation seems to depend on the input very much.

But:

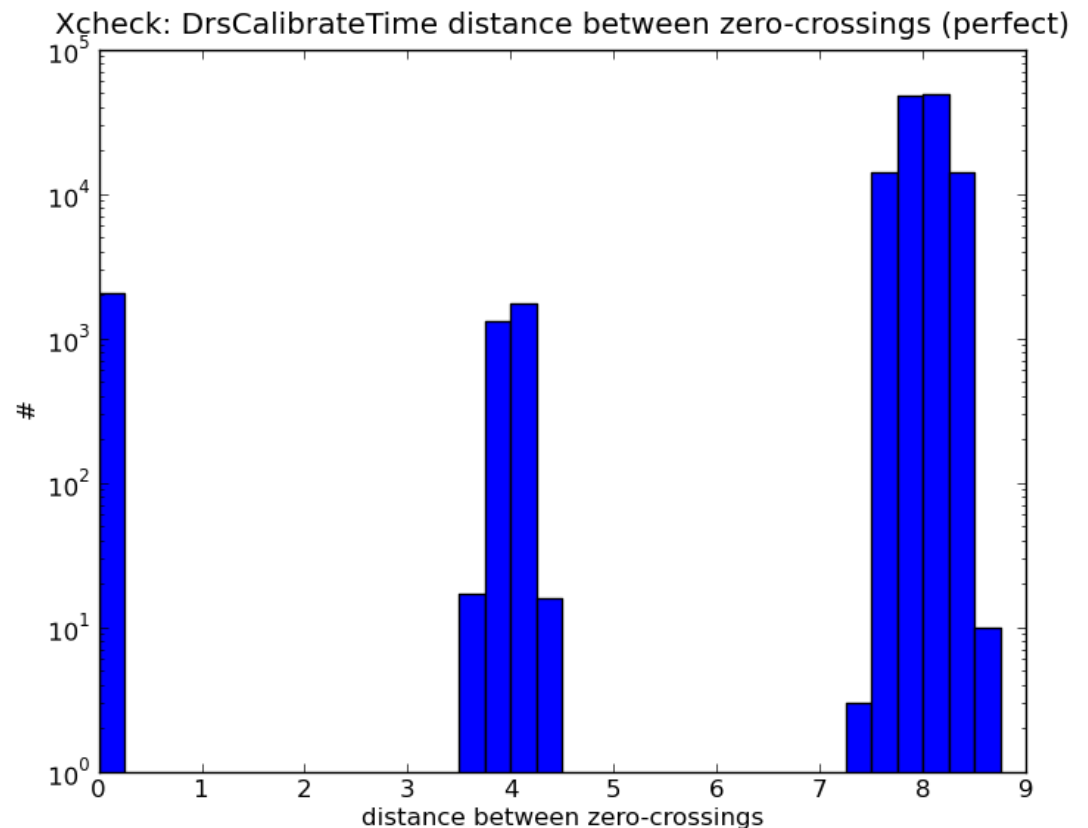
- I still don't understand, why
$$o_n = n \cdot (1 - s / s_n) \approx \sum_{i=0}^{n-1} d - d_i$$

Summary (I)

- I think our current implementation is working quite well.
- To Do:
 - Understand why it works.
 - Error estimation
 - Test influence of spikes:
 - undetected spikes might produce false zero crossings
 - detected and removed spikes might alter zero crossings

Deeper look into DrsCalibrateTime

- Check: Does DrsCalibrateTime find the right zero crossings?
 - At first glance: No (?!)
 - Reason:
When a sample == 0.
 - sample values are integers by nature, so 0 is not so unlikely. Only due to DRS calib we have floats and 0. samples are again unlikely.

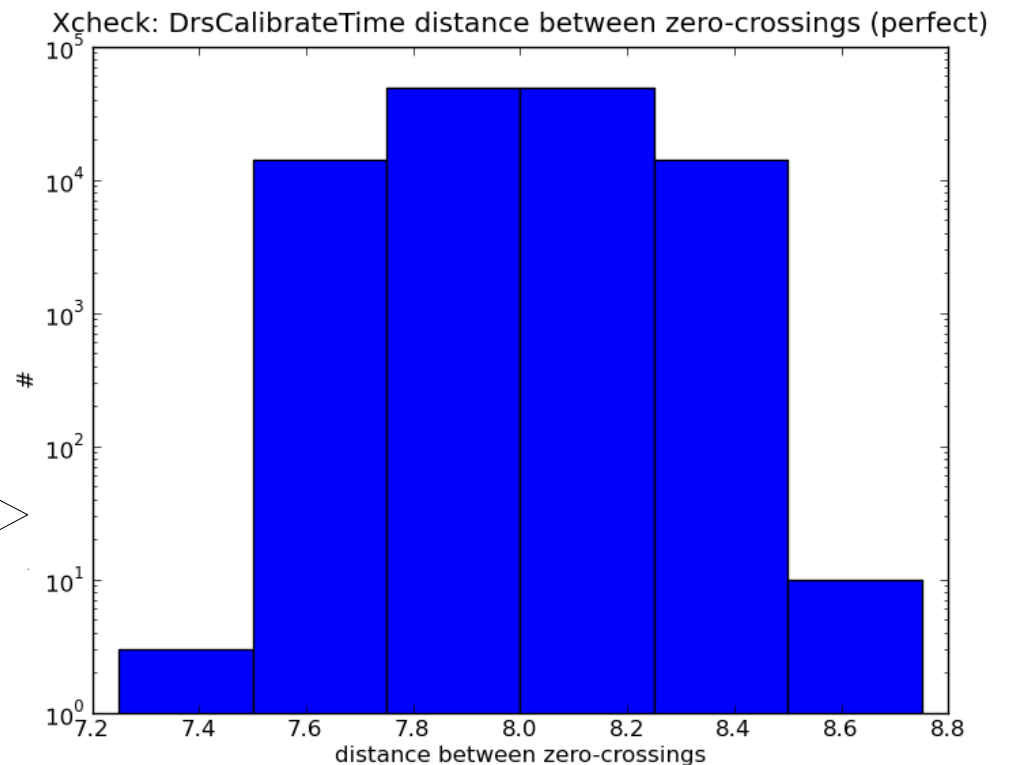
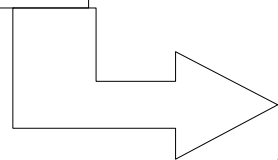


- So I changed this piece:

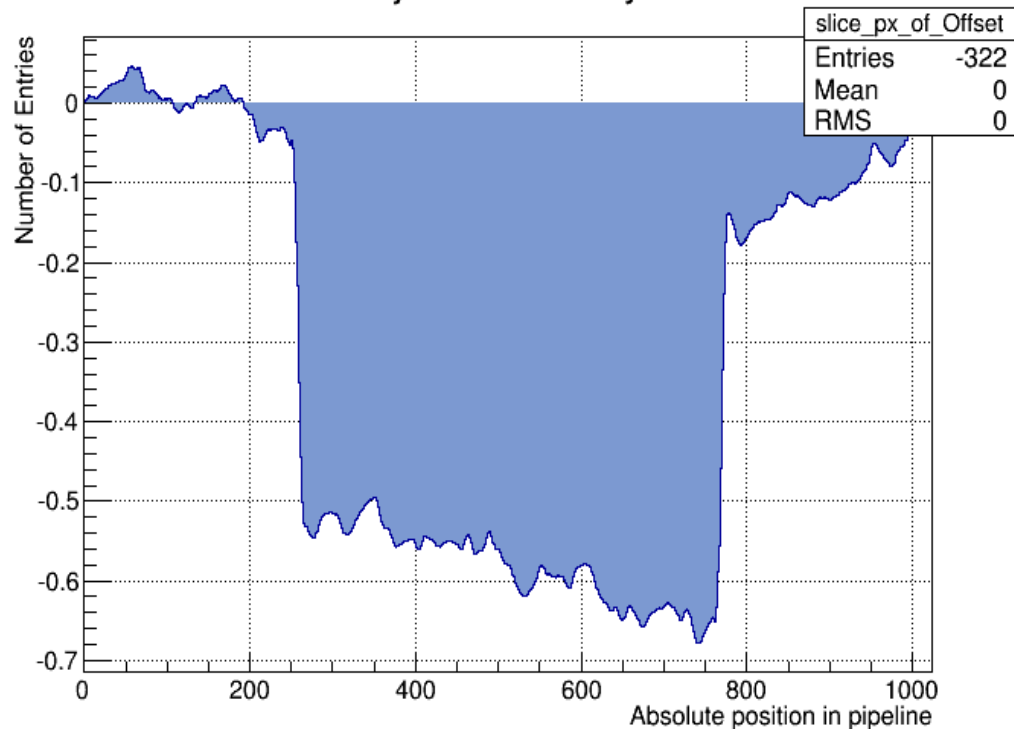
```
for (size_t i=0; i<1024-1; i++) {  
    if (edge>0 && v[i]>0)  
        continue;  
    if (edge<0 && v[i]<0)  
        continue;  
    if ((v[i]<0 && v[i+1]<0) || (v[i]>0 && v[i+1]>0))  
        continue;  
    double p = v[i]==v[i+1] ? 0.5 : v[i]/(v[i]-v[i+1]);  
    // ...  
}
```

- like this:

```
for (size_t i=0; i<1024-1; i++) {  
    if (v[i] * v[i+1] > 0.)  
        continue;  
    if ( edge * v[i+1] <= 0.)  
        continue;  
    const double p = v[i]/(v[i]-v[i+1]);  
    //...  
}
```



ProjectionX of biny=1



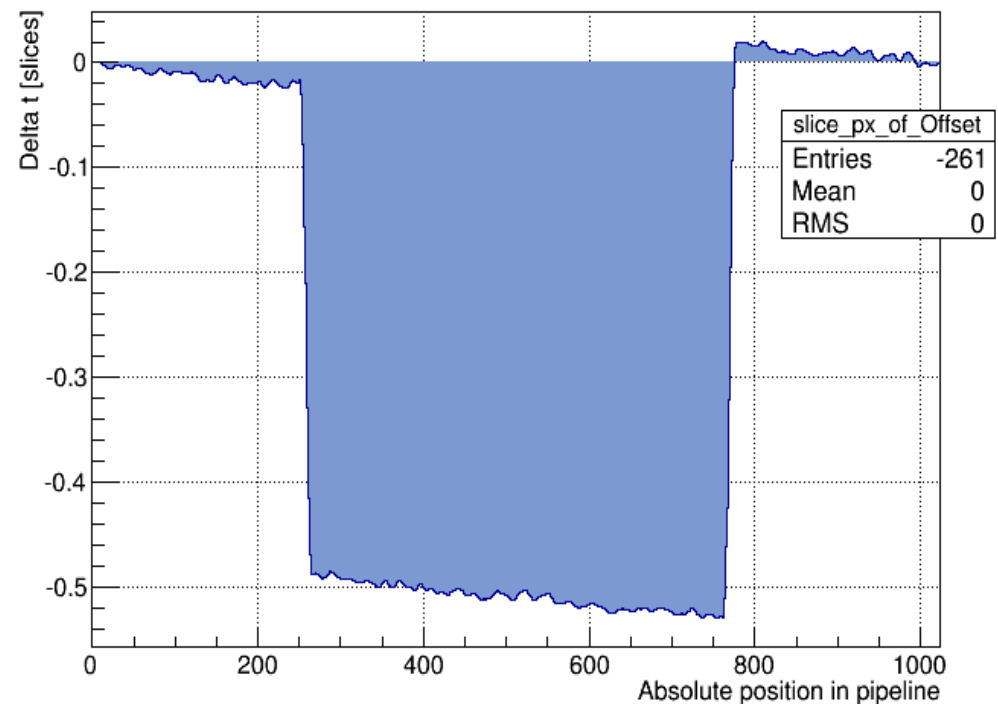
← Before

After →

Reminder:

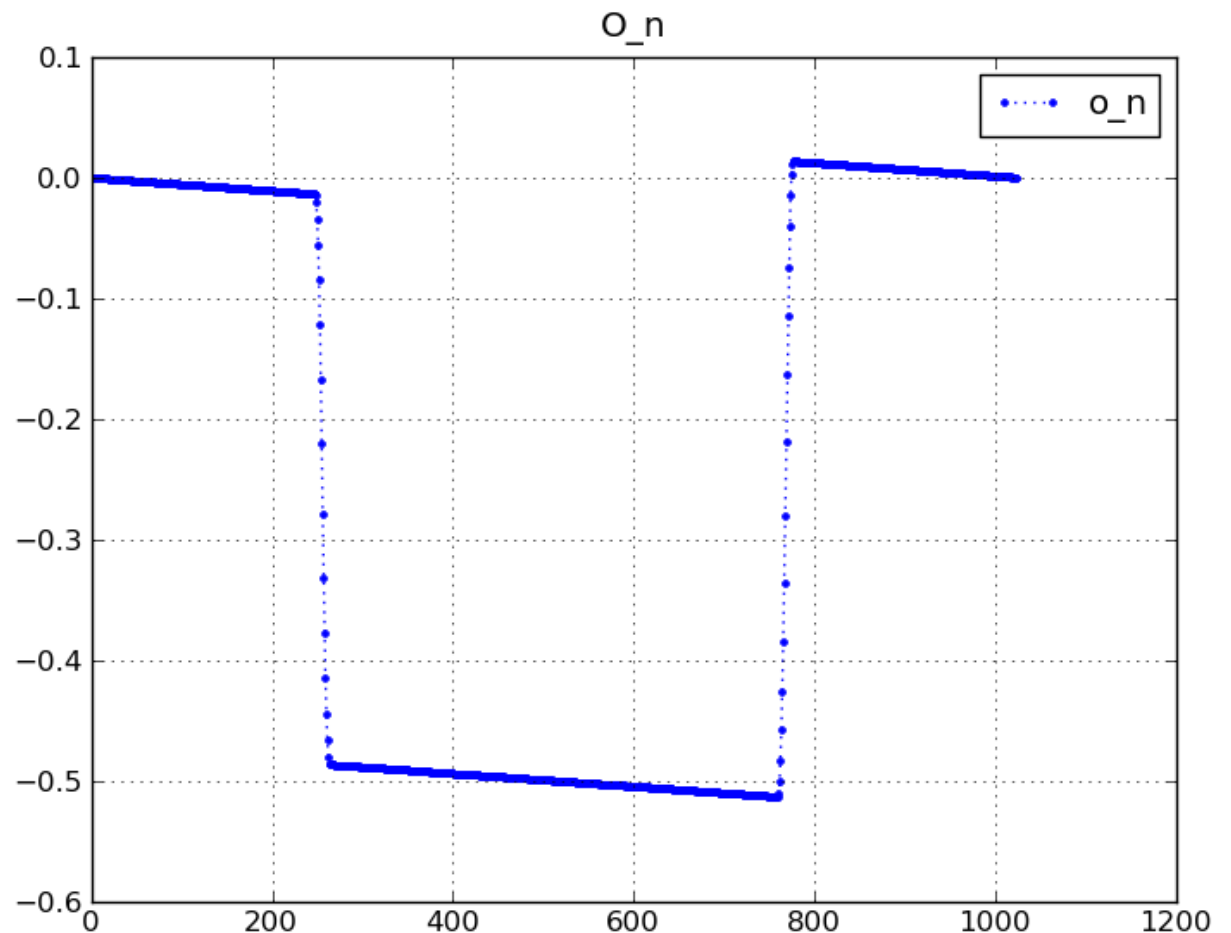
Since on real data, we hardly ever see the value 0.0, the performance on real data was actually more like the "After" than the "Before" picture.
So we were good!

INL : peak.fits

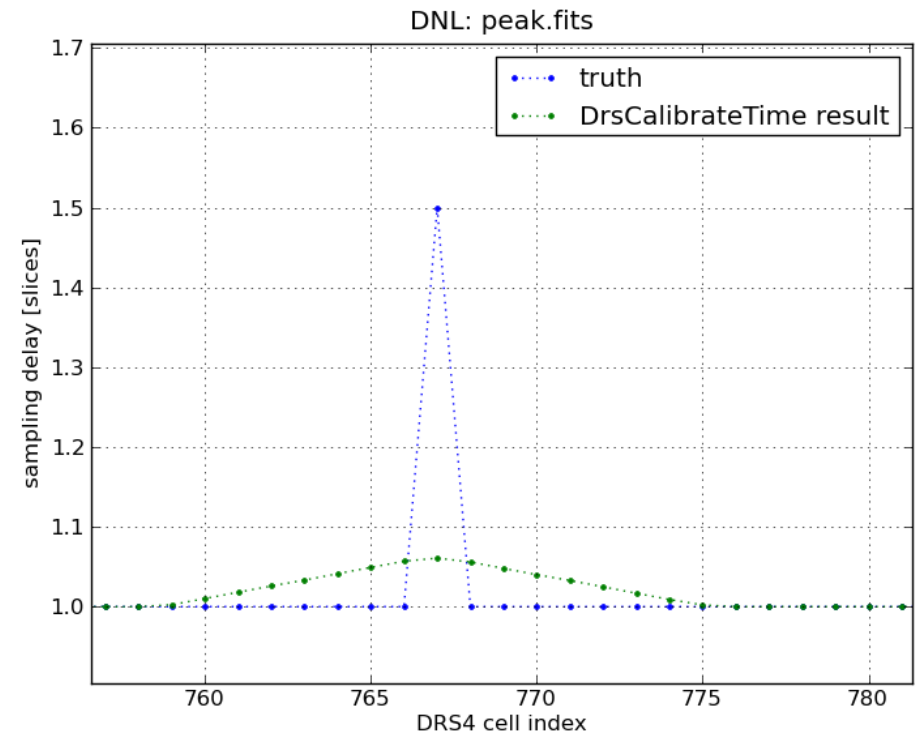
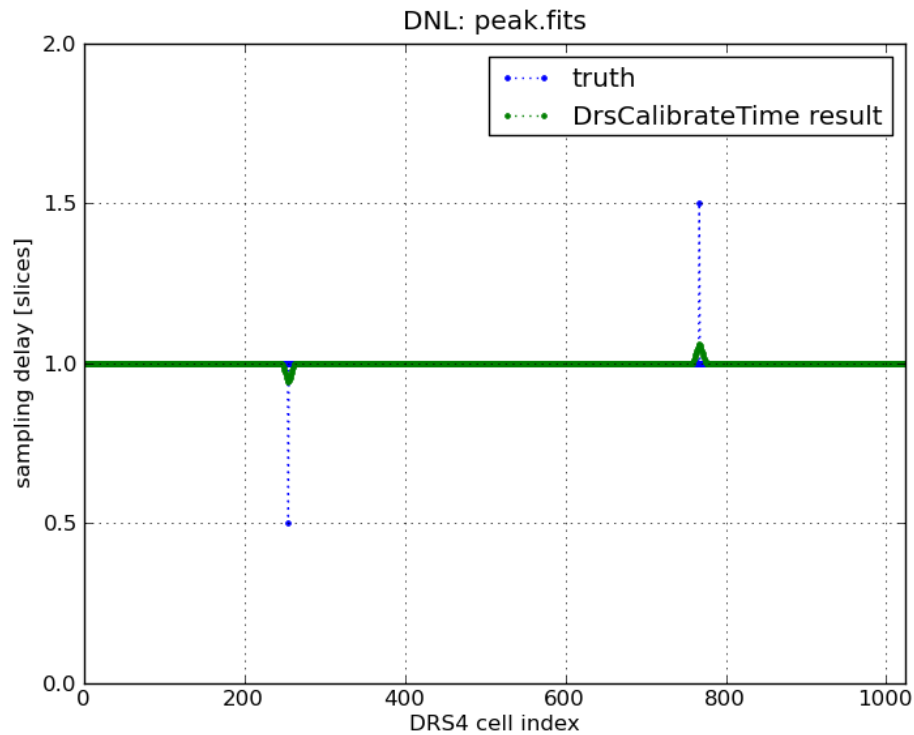


Best possible solution?

- I was curious, what the best solution would be, that I could get using DrsCalibrateTime.
- → feed the true zero crossings.



DNL : Compared to the truth



- Even under best conditions the truth can not be reconstructed with this method. Why is that so?
- Can we do better?

Maybe this is also a way to go:

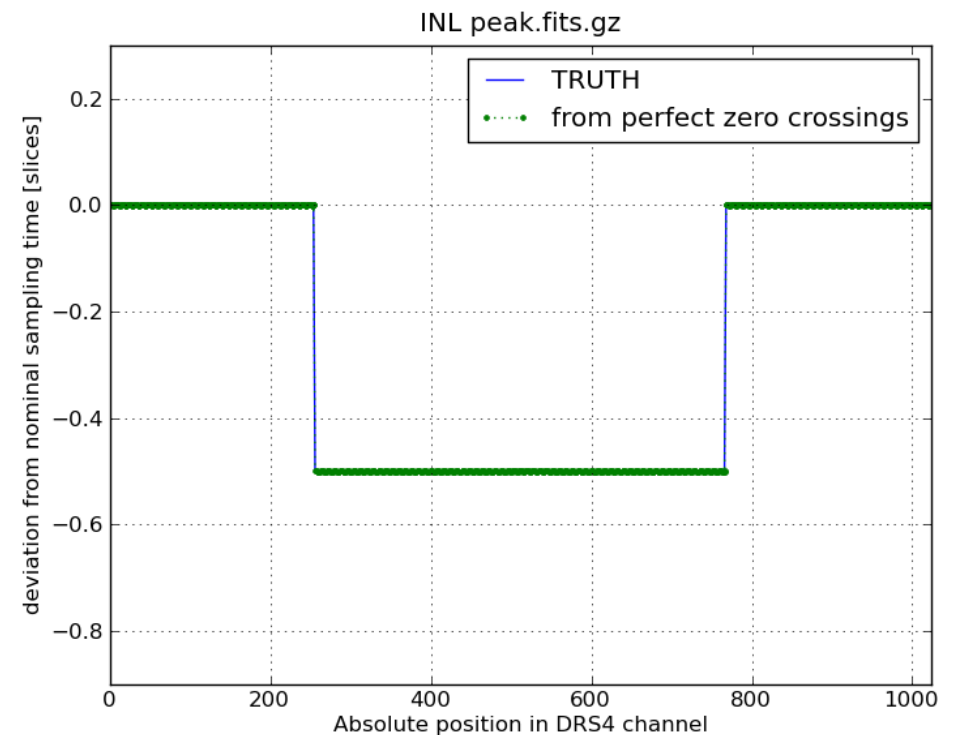
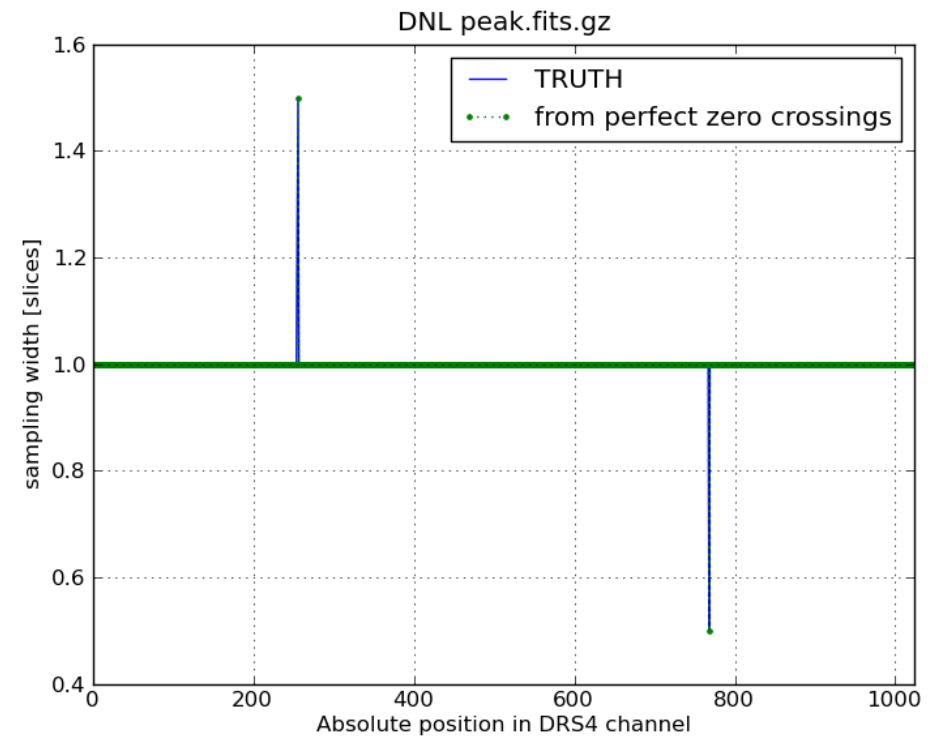
- Find the weights $w_{ik} \rightarrow$ Weight Matrix W
- $W(k \times 1024)$: with $k=M \cdot N$
 - M : number of periods per event ($\sim 2 \times 128$)
 - N : Number of events (~ 1000)
- Multiplication of this matrix with the sampling delay vector \mathbf{d} must yield the true calibration Period.
- The calibration Period is either known in advance or estimated as s_{1024} (see class DrsCalibrateTime)
- So we have to solve: $W \cdot \vec{d} = \vec{P}_{cal}$

Testing!

- I tested this method similarly, as the current implementation in `DrsCalibrateTime`.
- Result 1:
 - Data with const phase produces oscillating results.
- Result 2:
 - For data with variable phase the results look fine.
 - → let's look only at those results

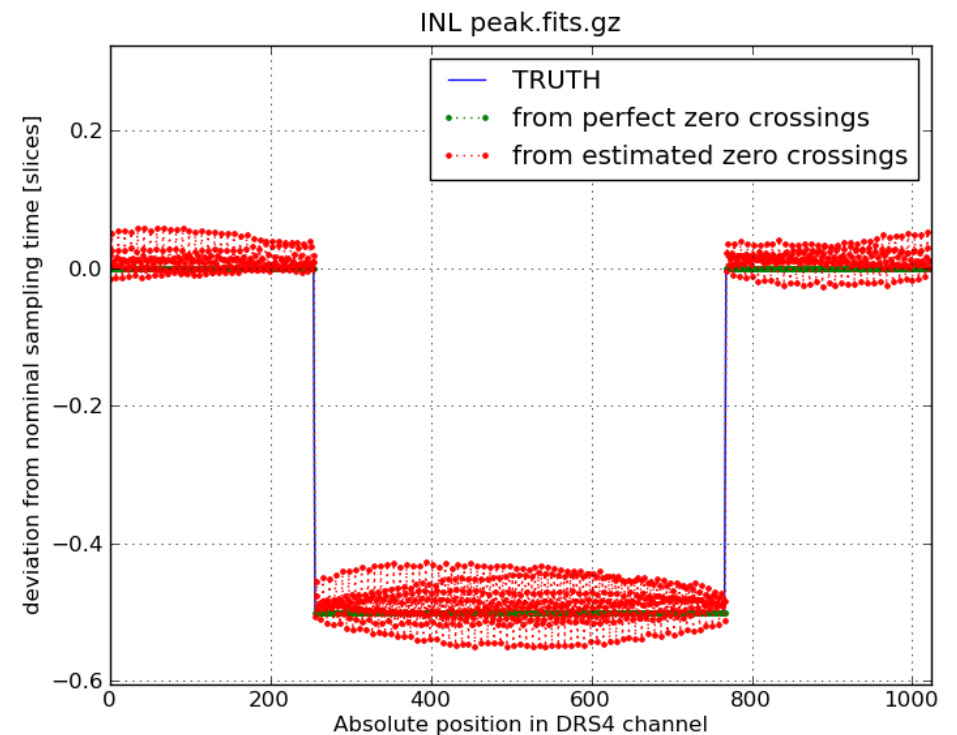
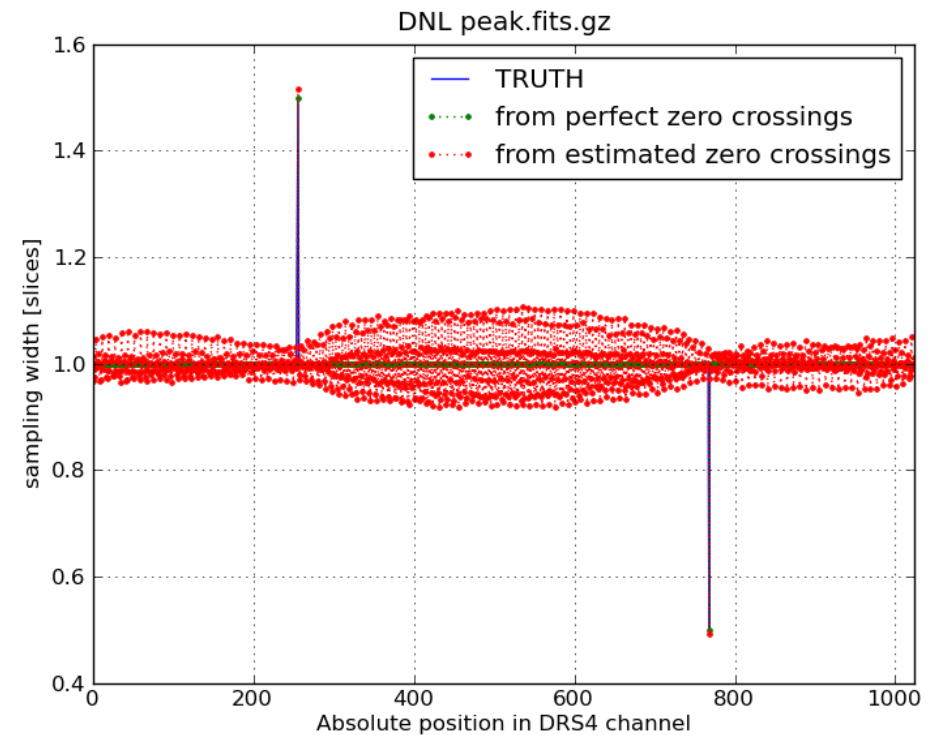
peak.fits

- In case the true zero crossings are used to find the w_{ik} the result is perfect.



peak.fits

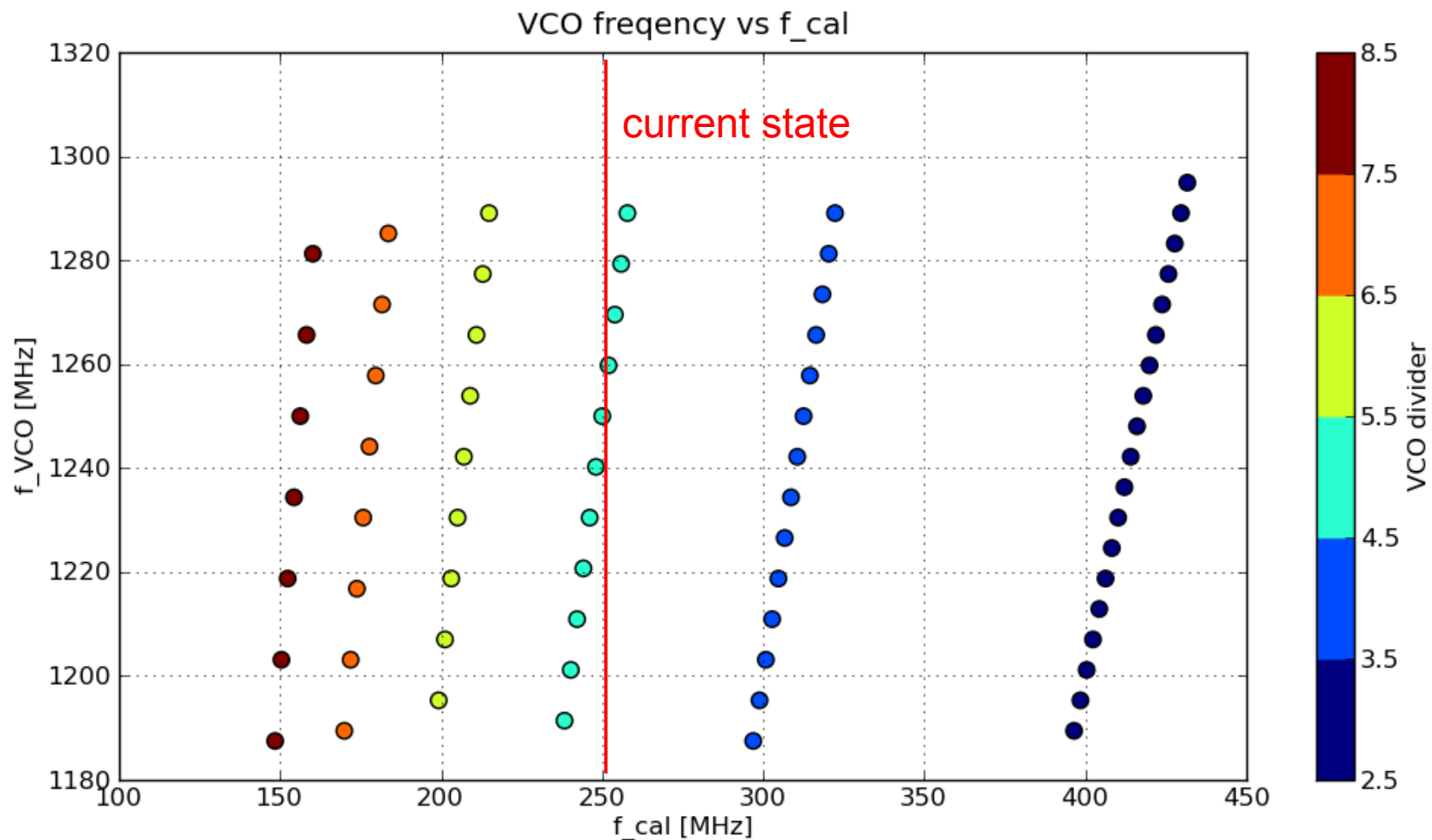
- In case the true zero crossings are used to find the w_{ik} the result is perfect.
- Oscillations:
 - due to the inaccurate measurement of the zero crossings?



- In order to use the 2nd method, we need a variable phase between calibration signal and domino wave.
- How to get this without touching the hardware?
 - Using the LMK03000 output delay:
17 different delays: [0, 400, 550, 700, ..., 2650]ps
 - Using different LMK03000 settings to get different calibration frequencies.
(currently we have 250MHz)

LMK03000

- 68 different possibilities
 $\text{DRS_REF_CLK} = 0.9765625\text{MHz}$



Summary (II)

- I think our current implementation is working quite well, but we may want to have a look into the 2nd method, I presented here.
- Ritt et al have shown a new method at IEEE, but I have so far not seen any slide or any detail about it.
- It might be worth to take "a whole lot" of different time calibration data once, and see what we can learn from it.