LA-UR-15-24590

Approved for public release; distribution is unlimited.

Title:	Algorithm for Beam Position and Phase Monitors in the LANSCE Linac
Author(s):	Mccrady, Rodney Craig
Intended for:	To share the information with colleagues at other accelerator labs
Issued:	2015-06-18

Algorithm for Beam Position and Phase Monitors in the LANSCE Linac

Rod McCrady

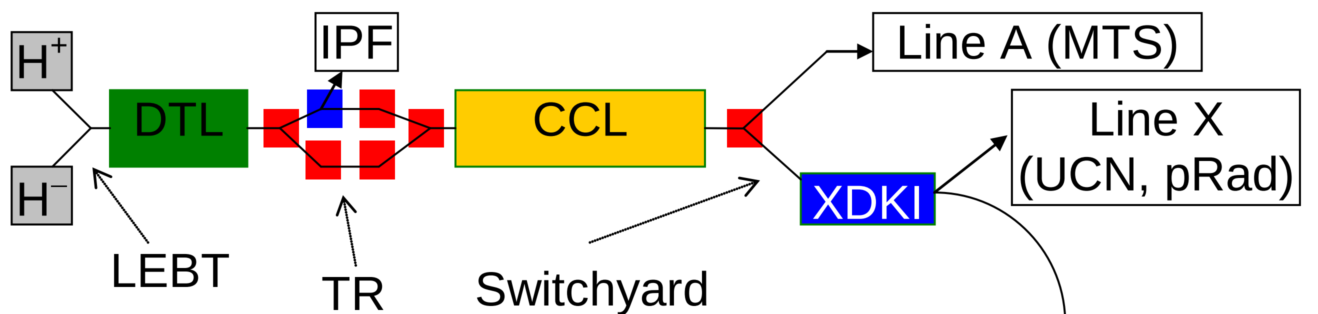
Outline

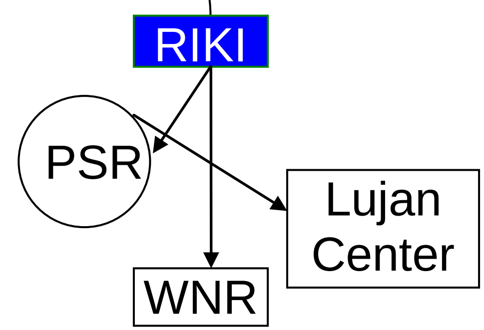
- Introduction: LANSCE and the BPPM system
- Initial algorithm and its shortcomings
- Improved algorithm
- Details of implementation
- Comparison to another algorithm
- Signal processing modes
- Performance
- Summary

3 levels of time structure

longer time ⟷ shorter time

macropulse	minipulse	micropulse
1ms	100ns	1ns
accelerator on/off	beam pulsed	RF acceleration





2 types of beam ions (H^+, H^-)

5 experiment areas

3 pulse formats / BPM processing modes

Micropulses

Micropulses result from RF acceleration

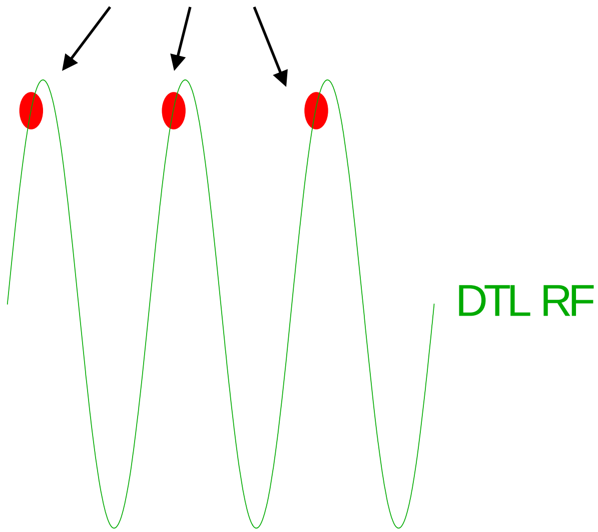
- 201.25 MHz repetition rate

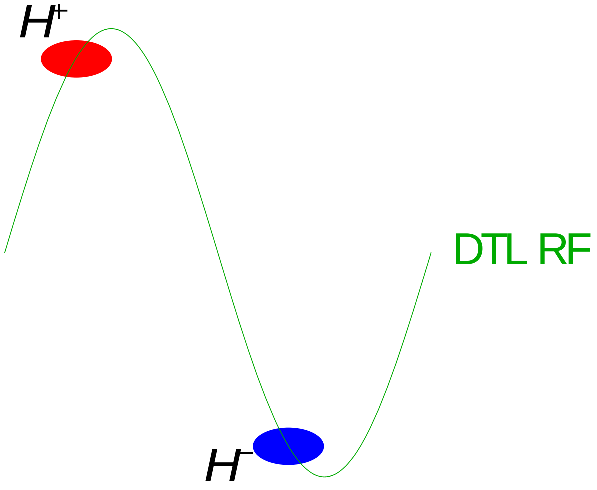
- 5pC to 125pC per micropulse

- About 100ps long

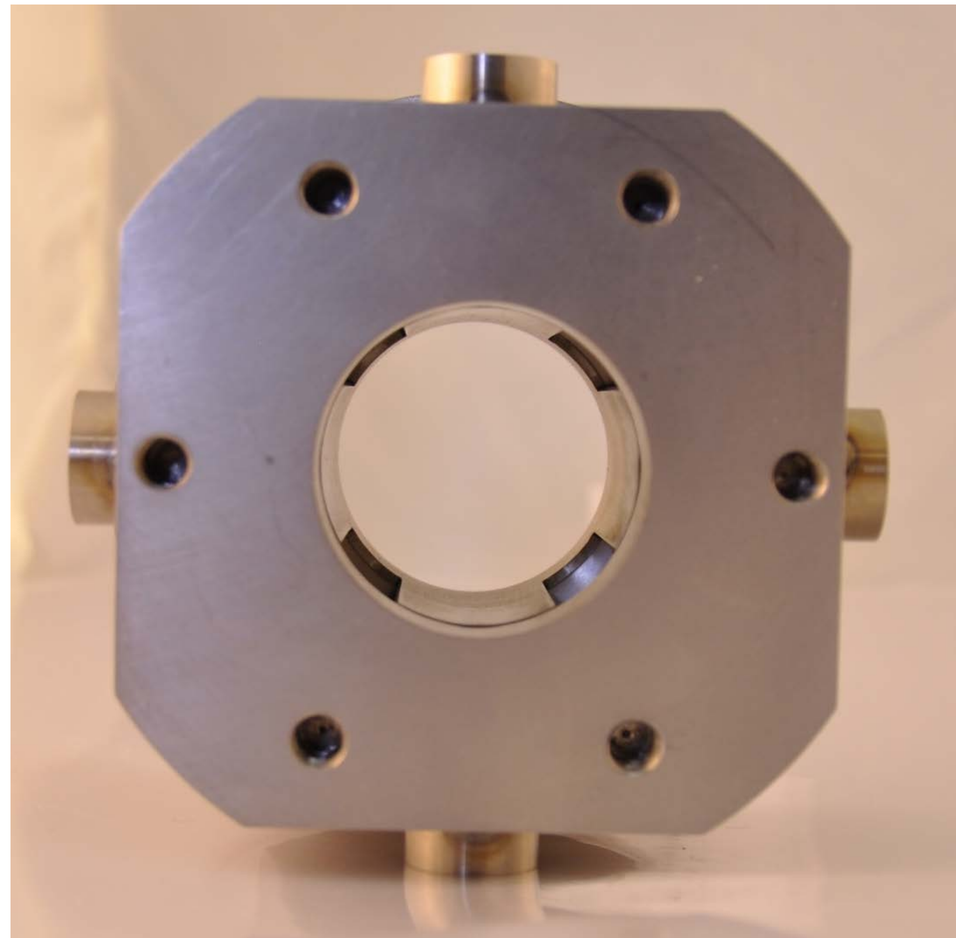
This is the source of the 201.25 MHz RF signals for BPPMs

Beam micropulses f = 201.25MHz ~100ps long

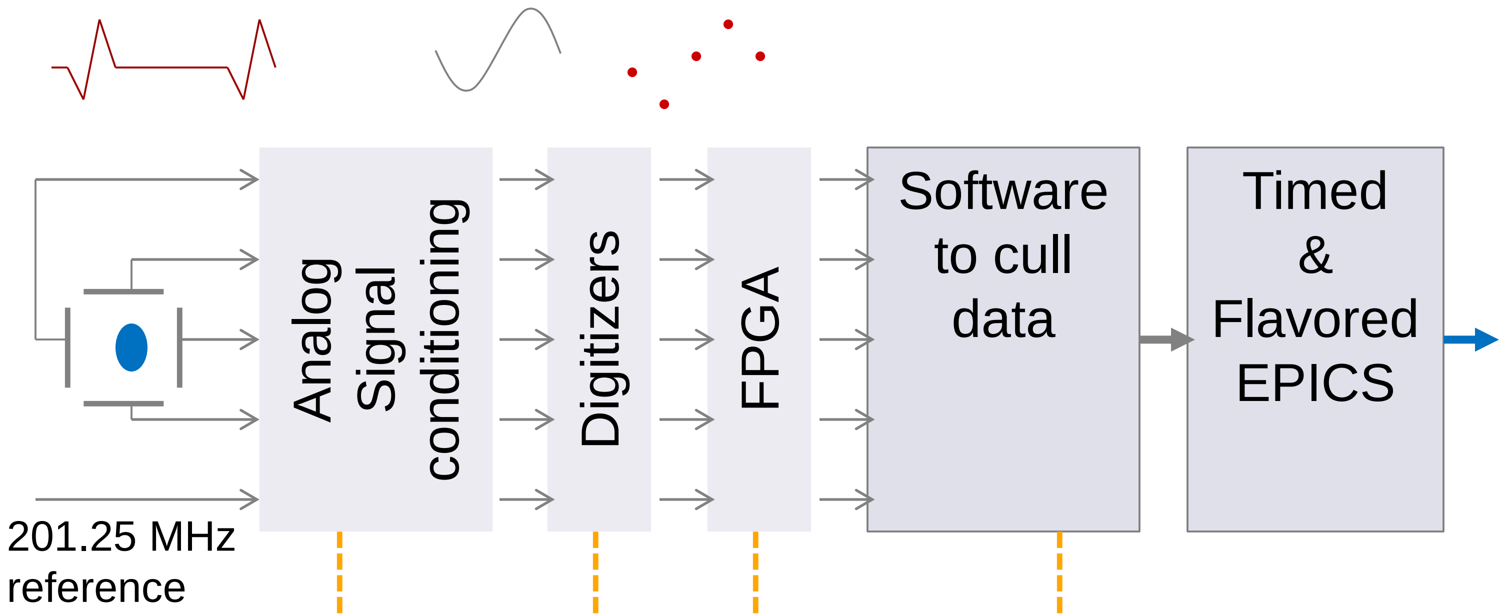




Transducers – origin of RF signals



Overview of the system



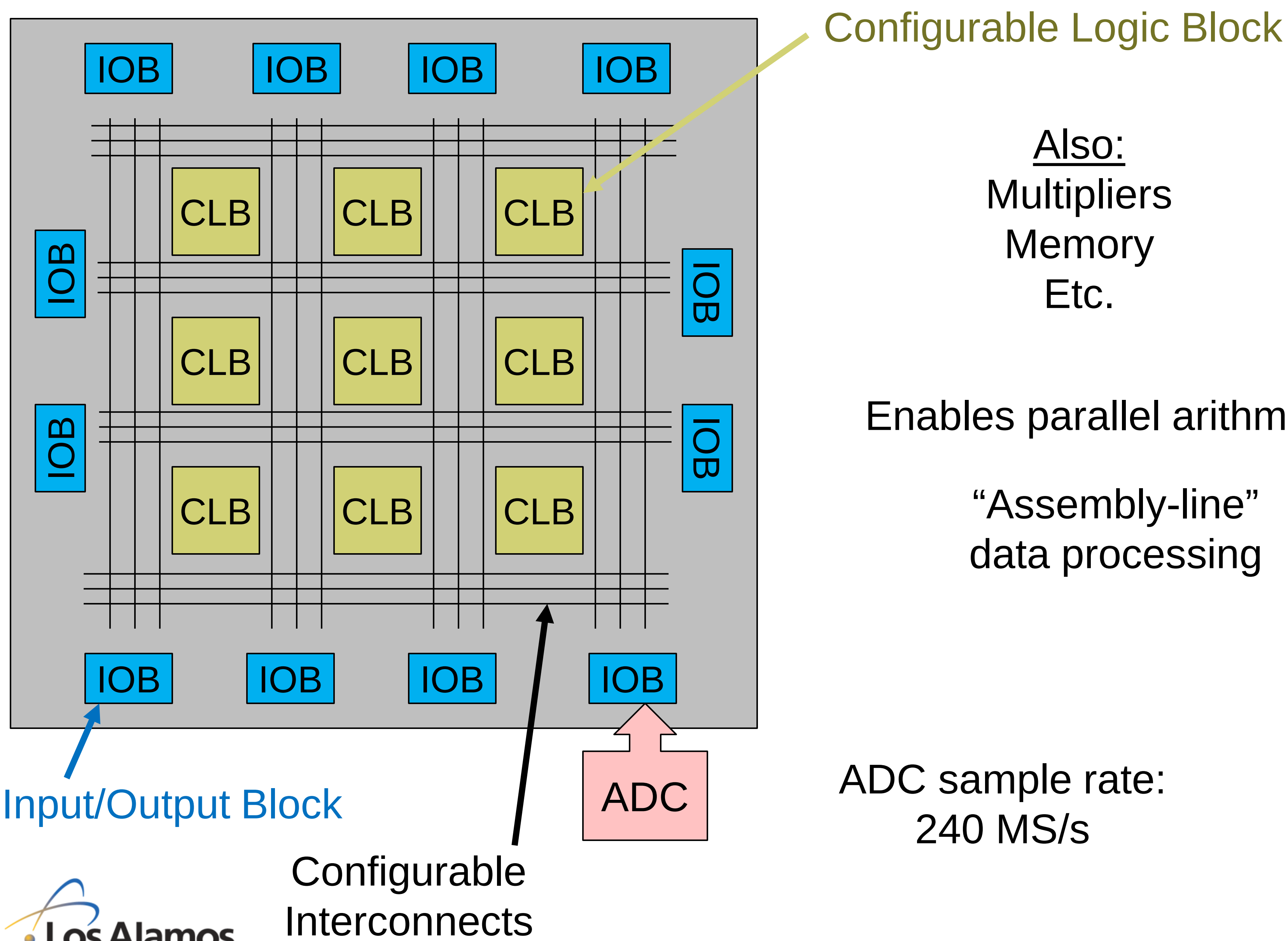
Filter
Amplifiers
Ringing filter (MPEG)
Cal. & diagnostics

Sinusoids
→H, V, ϕ, etc.
10k / gate

Undersampled
500kS / gate

Raw samples
Time mode
Successive mode
Gate-stacked mode

FPGA



Also:
Multipliers
Memory
Etc.

Enables parallel arithmetic

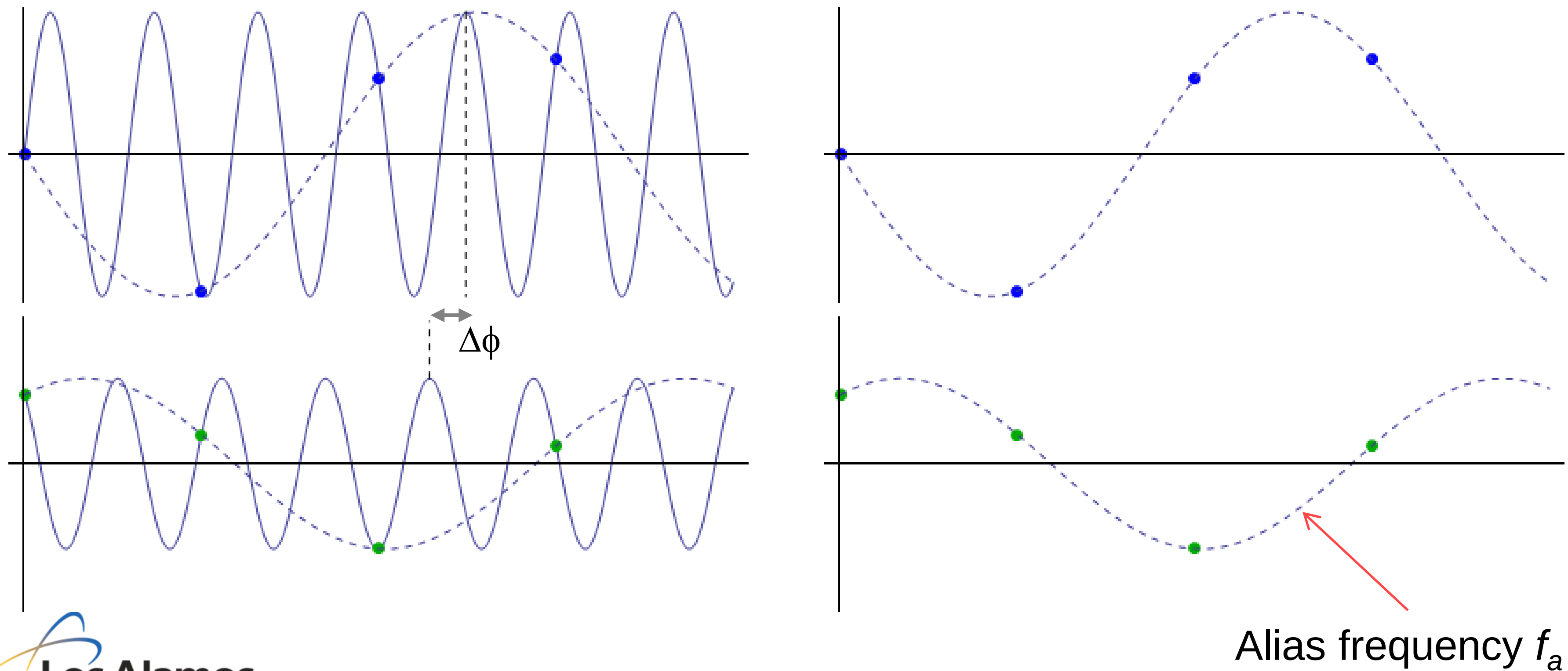
"Assembly-line"
data processing

ADC sample rate:
240 MS/s

- Introduction: LANSCE and the BPPM system
- **Initial algorithm and its shortcomings**
- Improved algorithm
- Details of implementation
- Comparison to another algorithm
- Signal processing modes
- Performance
- Summary

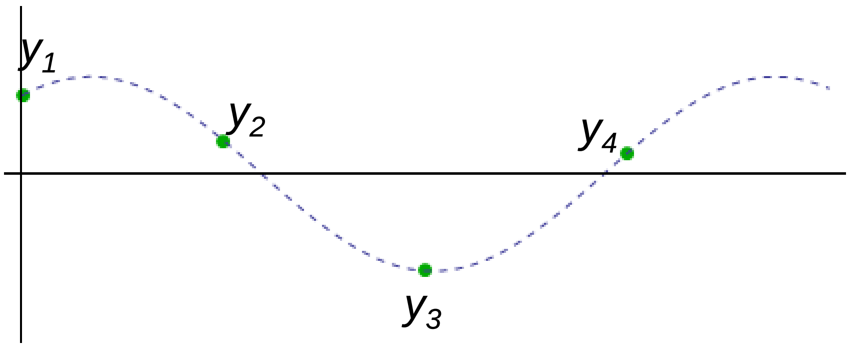
The problem

- **Suppose I digitize a test signal and a reference signal**
  - Both are sinusoids
  - The fundamental frequency is well-known and is identical for the two
  - The sample frequency is known *pretty well* (more on this later…)
- **I want to know the amplitude and phase of the test signal relative to the reference signal**



I&Q demod (sinusoid fit) with known frequencies

- **For now, assume the RF and sampling frequencies are known precisely**
- **Measure A & ϕ of the reference and test signals separately**



$$y_i = A\cos(wi - \phi) + y_0$$
$$= a\cos wi + b\sin wi + y_0$$

(just a trig identity)

where: $a = A\cos\phi$ and $b = A\sin\phi$

so

$\tan\phi = b/a$ and $A^2 = a^2 + b^2$

Note that *a* and *b* are I&Q

The samples:

$$y_i = A\cos(wi - \phi) + y_0$$

i : sample index

w : aliased frequency × Δt_{sample}

Parameters to determine:

A: amplitude

ϕ: phase

y_0 : DC offset

…I&Q demod (sinusoid fit) with known frequencies

The equation for the samples can be written as a matrix equation:

$$y_i = a\cos wi + b\sin wi + y_0$$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} \cos 1w & \sin 1w & 1 \\ \cos 2w & \sin 2w & 1 \\ \cos 3w & \sin 3w & 1 \\ \vdots & \vdots & \vdots \\ \cos Nw & \sin Nw & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ y_0 \end{pmatrix}$$

This is of the form *y=Mx*

where *y* and *M* are known and *x* is unknown

(computed)

(measured)

which can be solved using singular value decomposition or other well-known techniques

$$M : N_{samples} \times 3$$

But for implementation in and FPGA, I want a simpler, deterministic method for the solution.

The following technique only requires several multiplications and additions

…I&Q demod (sinusoid fit) with known frequencies

I'll define three vectors, *c*, *s*, and *u* :

$$c_i = \cos wi$$
$$s_i = \sin wi$$
$$u_i = 1$$

These can be computed in advance and stored in the FPGA

(The *u* vector may seem pointless, but it keeps the math neater)

The length of these vectors is the same as the data stream length

Now multiply each of these vectors by the equation for the samples:

$$y_i = a\cos wi + b\sin wi + y_0$$
$$\vec{y} = a\vec{c} + b\vec{s} + y_0\vec{u}$$

$$\vec{c}\cdot\vec{y} = a\vec{c}\cdot\vec{c} + b\vec{c}\cdot\vec{s} + y_0\vec{c}\cdot\vec{u}$$
$$\vec{s}\cdot\vec{y} = a\vec{s}\cdot\vec{c} + b\vec{s}\cdot\vec{s} + y_0\vec{s}\cdot\vec{u}$$
$$\vec{u}\cdot\vec{y} = a\vec{u}\cdot\vec{c} + b\vec{u}\cdot\vec{s} + y_0\vec{u}\cdot\vec{u}$$

Each dot-product is a scalar number

The three dot-product equations can be written as a matrix equation:

$$\begin{pmatrix} c\cdot y \\ s\cdot y \\ u\cdot y \end{pmatrix} = \begin{pmatrix} c\cdot c & c\cdot s & c\cdot u \\ s\cdot c & s\cdot s & s\cdot u \\ u\cdot c & u\cdot s & u\cdot u \end{pmatrix} \begin{pmatrix} a \\ b \\ y_0 \end{pmatrix}$$

Whose solution is:

$$\begin{pmatrix} a \\ b \\ y_0 \end{pmatrix} = \begin{pmatrix} c\cdot c & c\cdot s & c\cdot u \\ s\cdot c & s\cdot s & s\cdot u \\ u\cdot c & u\cdot s & u\cdot u \end{pmatrix}^{-1} \begin{pmatrix} c\cdot y \\ s\cdot y \\ u\cdot y \end{pmatrix}$$

…I&Q demod (sinusoid fit) with known frequencies

The solution:

$$\begin{pmatrix} a \\ b \\ y_0 \end{pmatrix} = \begin{pmatrix} c \cdot c & c \cdot s & c \cdot u \\ s \cdot c & s \cdot s & s \cdot u \\ u \cdot c & u \cdot s & u \cdot u \end{pmatrix}^{-1} \begin{pmatrix} c \cdot y \\ s \cdot y \\ u \cdot y \end{pmatrix}$$

These depend on the data and must be computed for each data acquisition

These don't depend on the data, but do depend on the data stream length

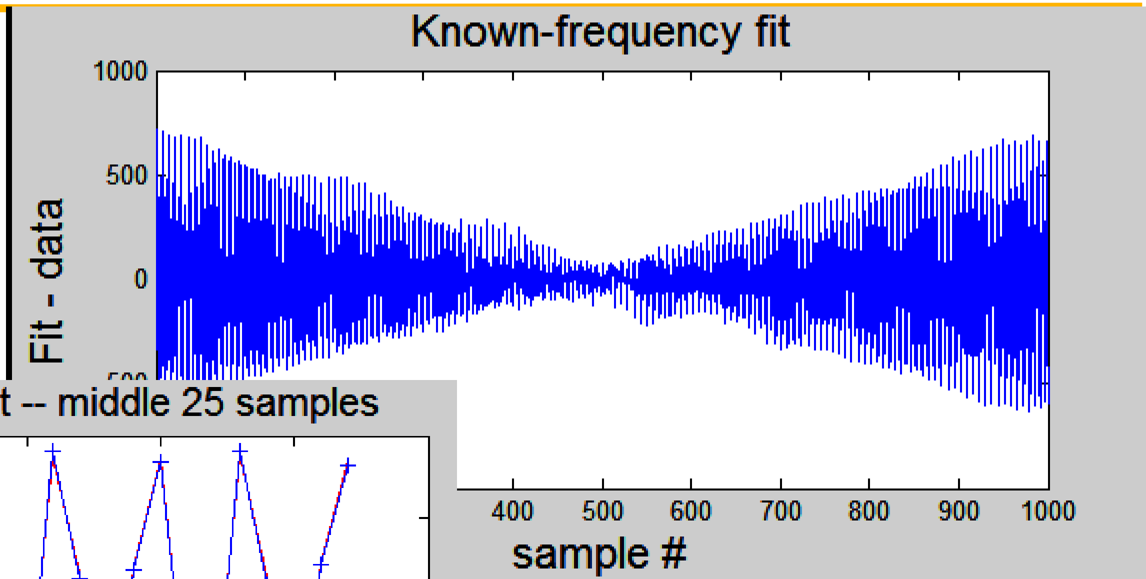
What if the alias frequency is not known precisely?

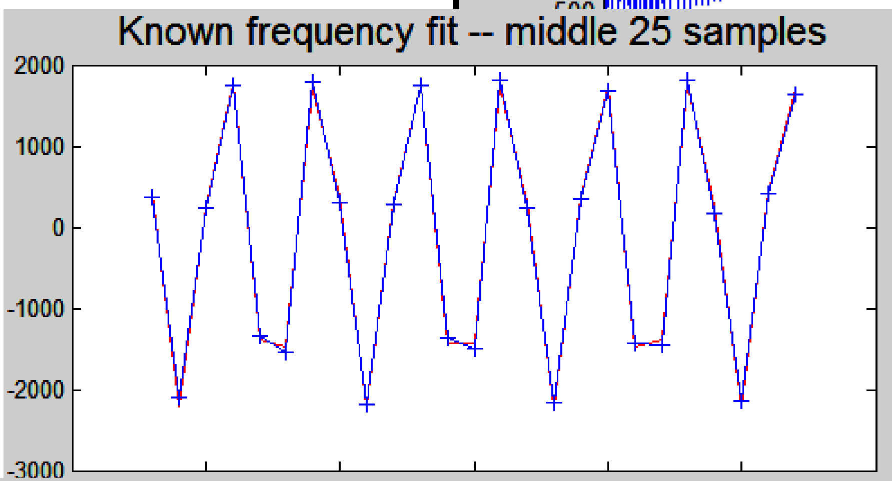
RF and sample clock aren't locked
$\rightarrow$ alias frequency will drift

A 1000 point fit, assuming

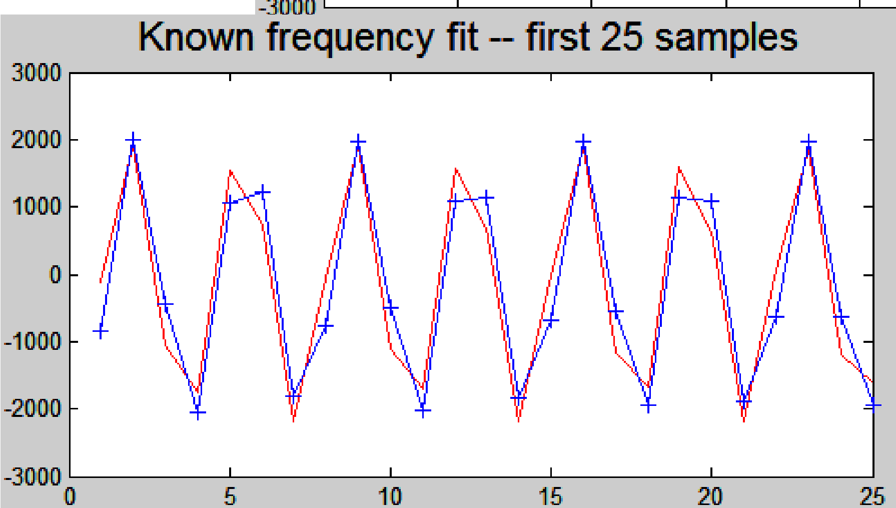
f_{RF} = 201.25 MHz and

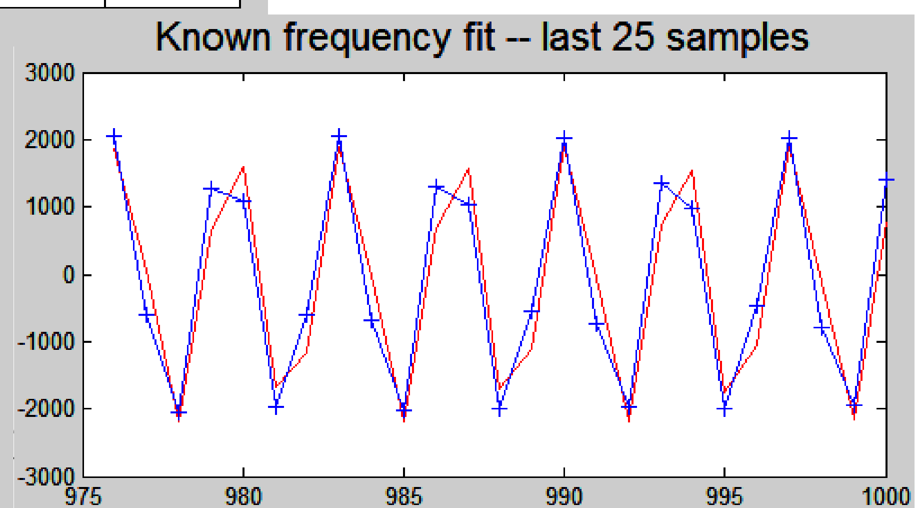
f_{sample} = 117.440 MHz



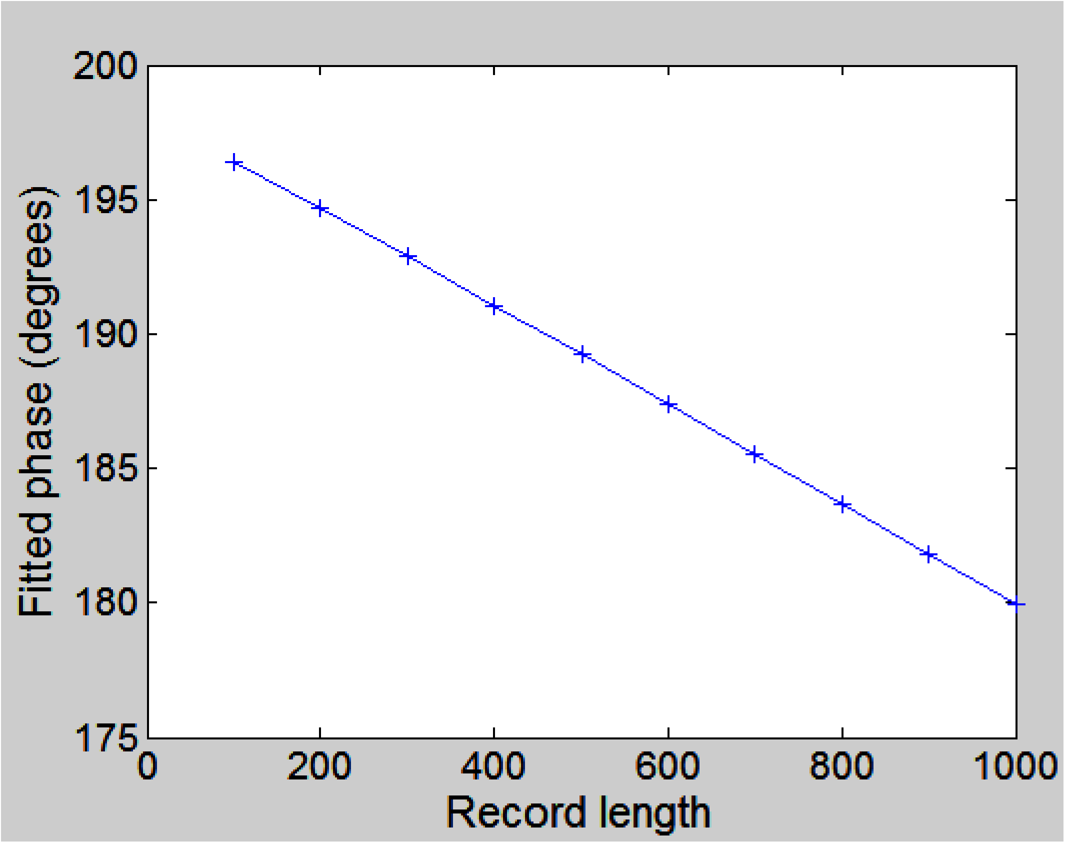


Why is fit better near the middle?





What if the alias frequency is not known precisely?



The assumed alias frequency was wrong

Fitted sample frequency: 117.446 MHz (51 ppm difference)

I want a method that is tolerant of small changes in sample frequency

- Introduction: LANSCE and the BPPM system
- Initial algorithm and its shortcomings
- **Improved algorithm**
- Details of implementation
- Comparison to another algorithm
- Signal processing modes
- Performance
- Summary

If the sample frequency is known *pretty well*

Rather than assuming a frequency when generating the vectors of sines and cosines:

$$\cancel{c_i = \cos wi \quad s_i = \sin wi}$$

Use the sampled reference instead

Sample *i* of the reference signal is: $r_i = R\cos(wi - \phi)$

Take the sampled reference to be the cosine vector

…but how about the sine vector?

The previous sample *i-1* of the reference signal is: $r_{i-1} = R\cos(w(i-1) - \phi)$

From these two samples I can get the sine

$$= R\cos(wi - \phi - w)$$

Expand

$$= \underline{R\cos(wi - \phi)}\cos w + R\sin(wi - \phi)\sin w$$

Trig identity

$$= \underline{r_i}\cos w + R\sin(wi - \phi)\sin w$$

Recognize r_i

$$\underline{R\sin(wi - \phi)} = \frac{r_{i-1} - r_i \cos w}{\sin w}$$

Solve to get sine term

This is the i^{th} element of the sine vector

…If the sample frequency is known *pretty well*

$$R\sin(wi-\phi)=\frac{r_{i-1}-r_i\cos w}{\sin w}$$

The values of cos$w$ and sin$w$ can be computed and stored.

Using this approach, the fitted waveform phase doesn't walk relative to the data.

Also, instead of fitting the test signal and the reference,

then subtracting the phases

(along with a few applications of the mod() function)

The phase of the test signal relative to the reference is obtained directly in the fit

…If the sample frequency is known *pretty well*

The fit algorithm is as described earlier, except the vectors of sines and cosines are computed:

$$c_i = r_i$$

$$s_i = \frac{r_{i-1} - r_i \cos w}{\sin w}$$

$$u_i = 1$$

This must be done for each data acquisition

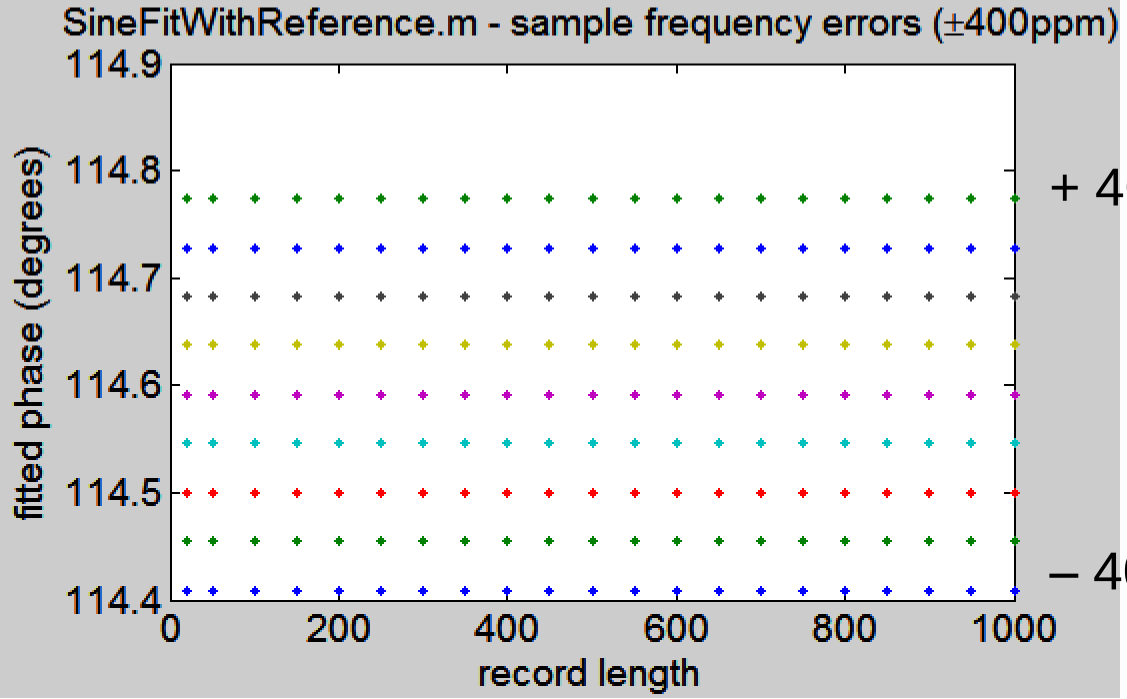
$$\begin{pmatrix} a \\ b \\ y_0 \end{pmatrix} = \begin{pmatrix} c \cdot c & c \cdot s & c \cdot u \\ s \cdot c & s \cdot s & s \cdot u \\ u \cdot c & u \cdot s & u \cdot u \end{pmatrix}^{-1} \begin{pmatrix} c \cdot y \\ s \cdot y \\ u \cdot y \end{pmatrix}$$

The inverse matrix can't be computed ahead of time and stored, but the dot products and 3x3 matrix inversion are straightforward arithmetic

Requires that there is no DC offset on the reference signal

Sensitivity to frequency errors

The constants cos*w* and sin*w* are computed for the assumed frequency.



+ 400 ppm

– 400 ppm

- Introduction: LANSCE and the BPPM system
- Initial algorithm and its shortcomings
- Improved algorithm
- **Details of implementation**
- Comparison to another algorithm
- Signal processing modes
- Performance
- Summary

Computing sine vector with integer arithmetic

sin$w$ and cos$w$ are ≤ 1

Problem for integer arithmetic

$$c_i = r_i$$

$$s_i = (r_{i-1} - r_i \times \cos w) \times \frac{1}{\sin w}$$

Multiply

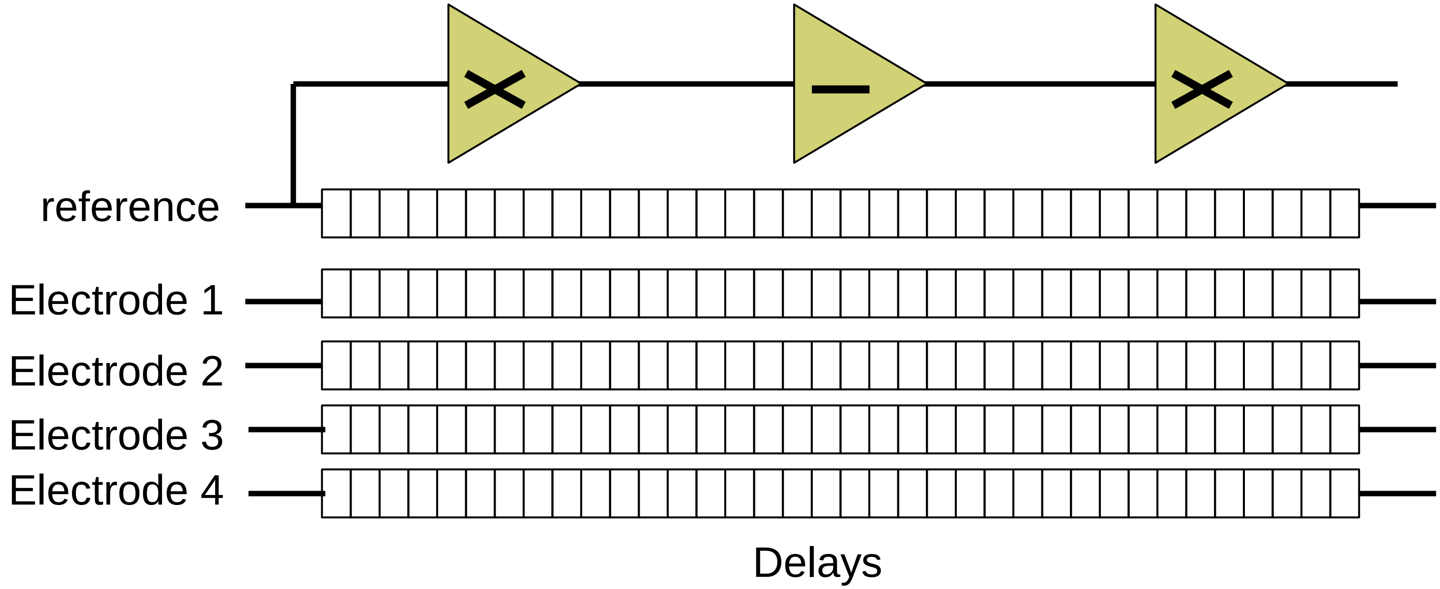
Subtract

Multiply

clock cycles

	Integer	Float
subtract	1	14
multiply	1	11

Floating-point arithmetic uses more logic, too



Computing sine vector with integer arithmetic

Ahead of time:

$$s_i = (r_{i-1} - r_i \times \cos w) \times \frac{1}{\sin w}$$

Store constants: $2^k \times \cos w \qquad \frac{2^k}{\sin w}$

Large integers → small round-off error

While processing:

Multiply *i-1* reference sample by 2^k
(append *k* zeros)

Multiply, subtract, multiply

Divide by 2^{2k}
(drop 2*k* bits)

$$s_i = \left\{ \left[2^k \times r_{i-1} - r_i \times (2^k \times \cos w) \right] \times \frac{2^k}{\sin w} \right\} \div 2^{2k}$$

Sine vector with integer arithmetic -- Choosing k

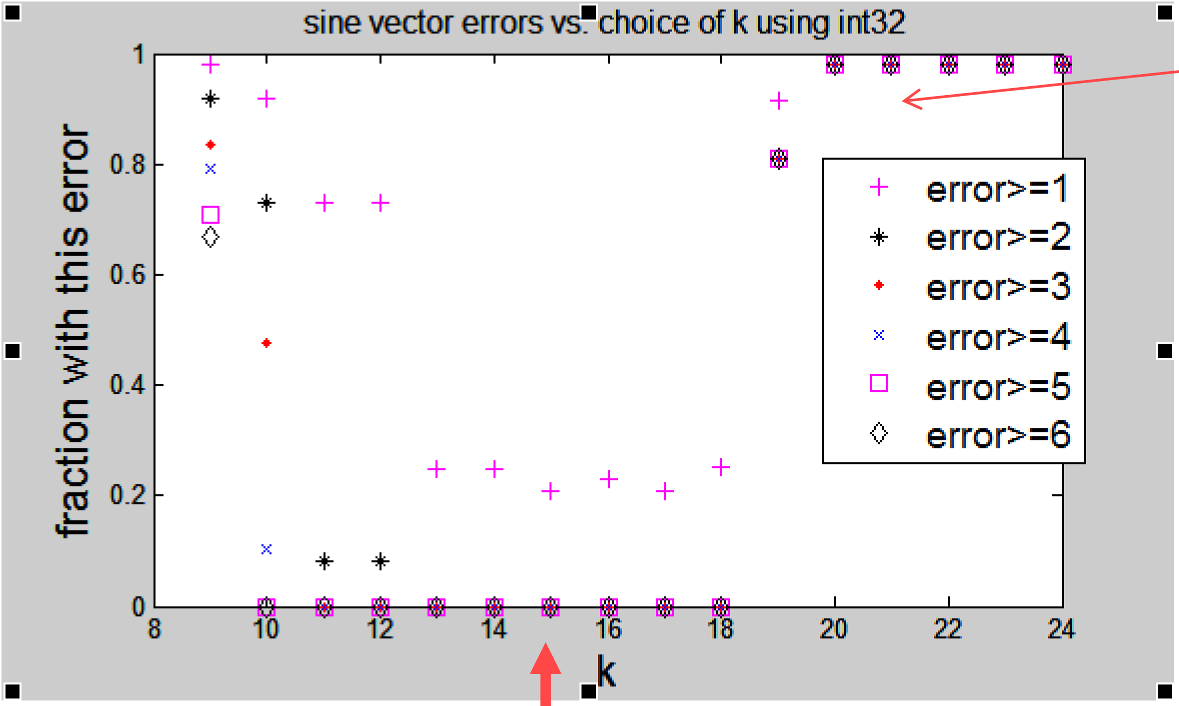
Using integers:

Compute *Acos*(*iw*), then *Asin*(*iw*) using our algorithm

Directly compute *Asin*(*iw*)

($A = 8192 = 2^{13}$)

Then compare these two #'s

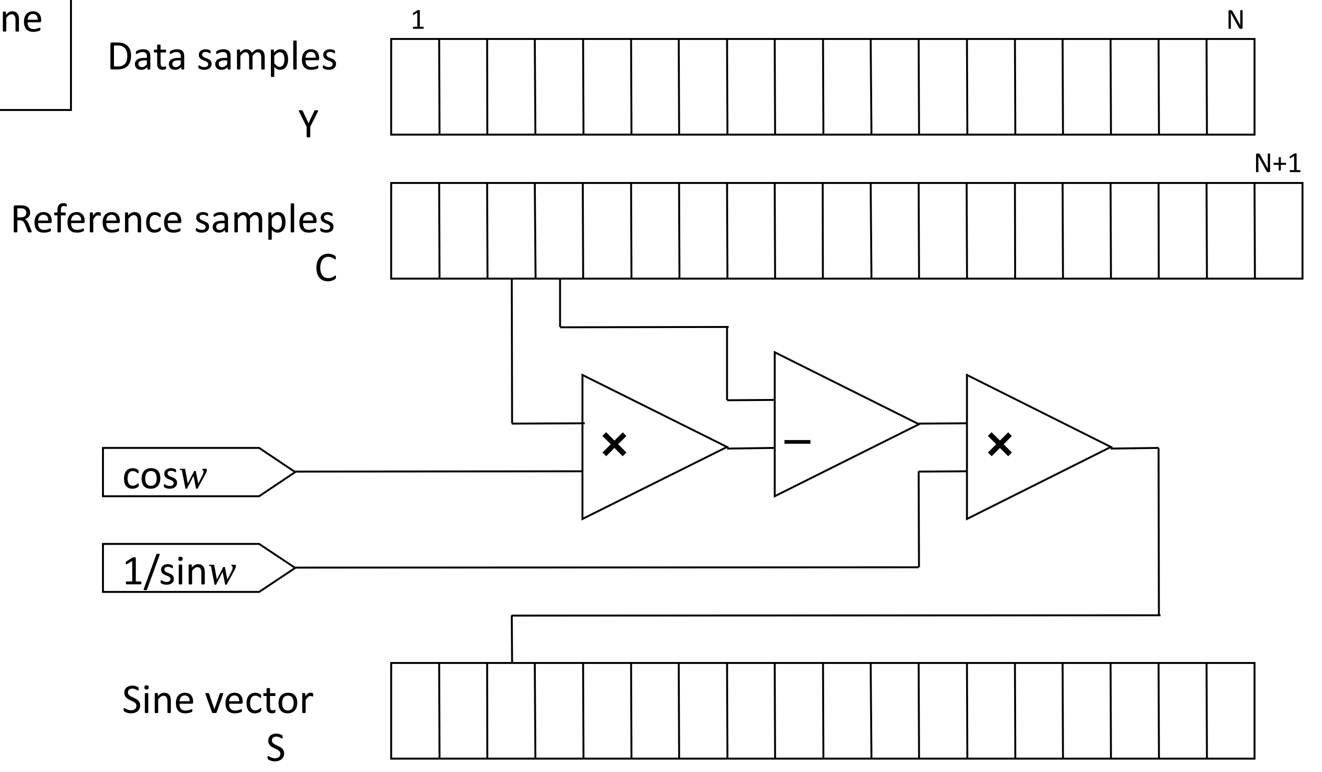


Overflows of 32-bit integers

k = 15

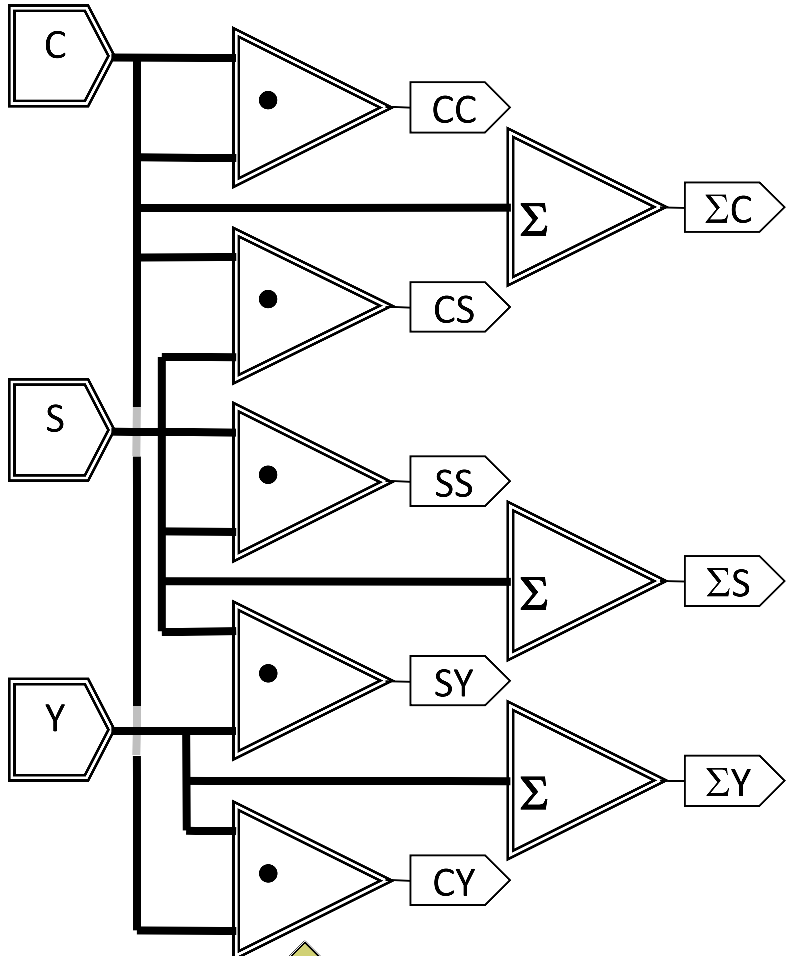
Block diagrams of arithmetic

Step 1:
Generate sine vector



Step 2:
Reduce vectors
to scalars

Data streams



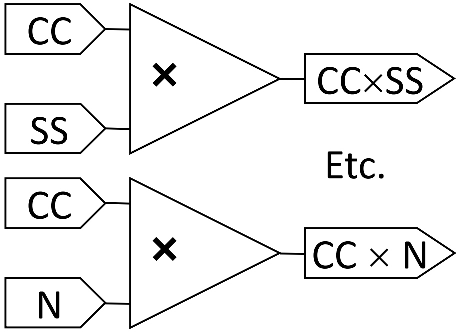
scalars

Electrode signals
(4 of these)

Sequencing
Logic

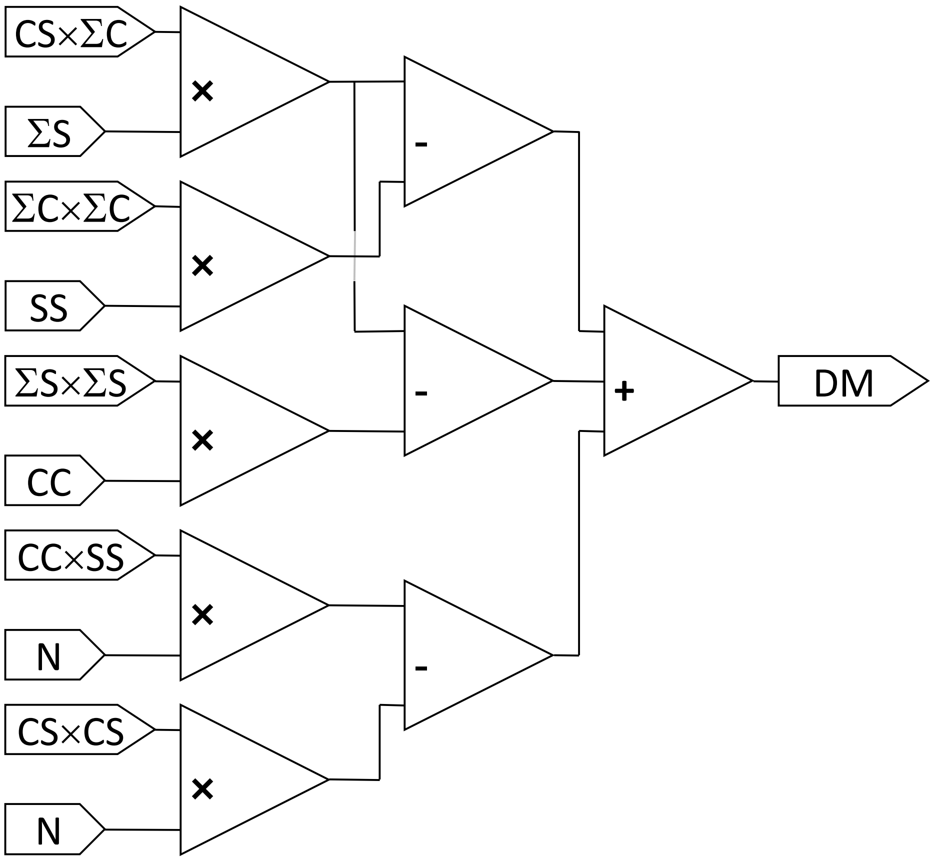
Convert these to
floating-point
values

Step 3:
Get products
of scalars

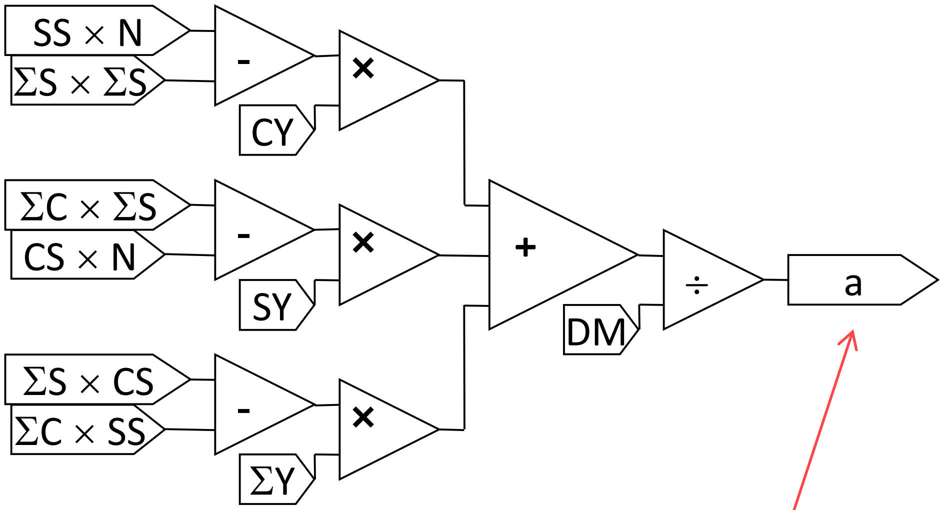


13 of these; pairs of:
CC, CS, SS, ΣC, ΣS, N

Step 4:
Get the determinant of a matrix



Step 5:
Get the final quantities of interest



$$y_i = a\cos wi + b\sin wi + y_0$$

3 blocks like this (a, b, y_0)
for each electrode

- Introduction: LANSCE and the BPPM system
- Initial algorithm and its shortcomings
- Improved algorithm
- Details of implementation
- **Comparison to another algorithm**
- Signal processing modes
- Performance
- Summary

Compare to Goertzel algorithm

Used for phase-control on other projects

Single-component Digital Fourier Transform (DFT) (but computed differently)

Computationally efficient

$$s[n] = y[n] + 2\cos\left(\frac{2\pi k}{N}\right)s[n-1] - s[n-2]$$

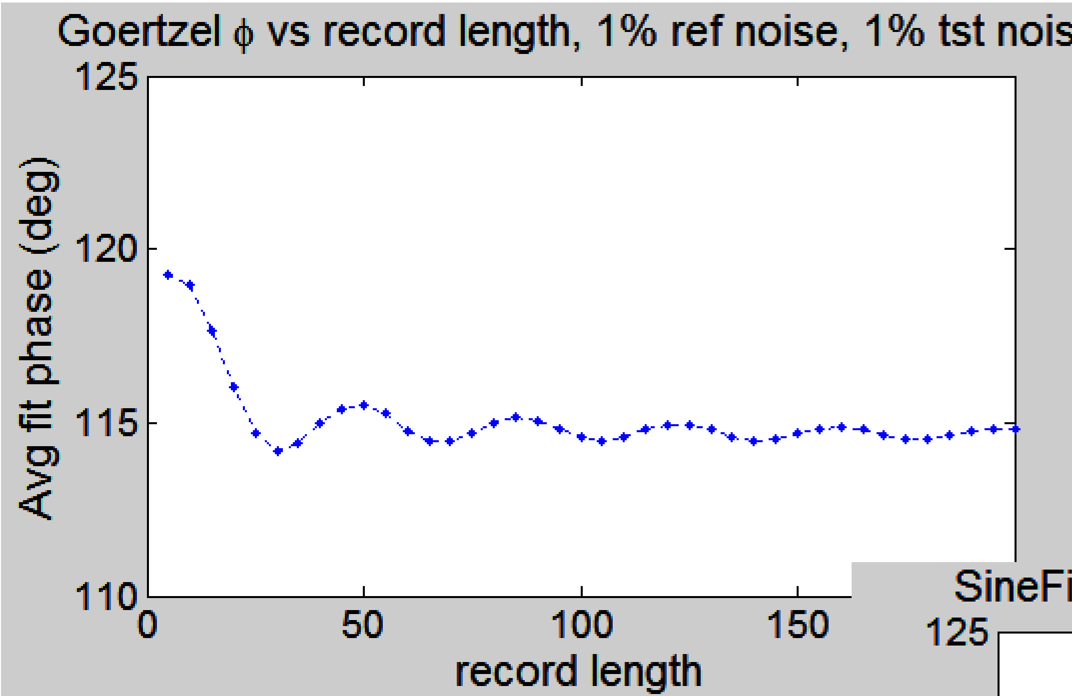
$$x_k[n] = s[n] - \exp\left(-i\frac{2\pi k}{N}\right)s[n-1]$$

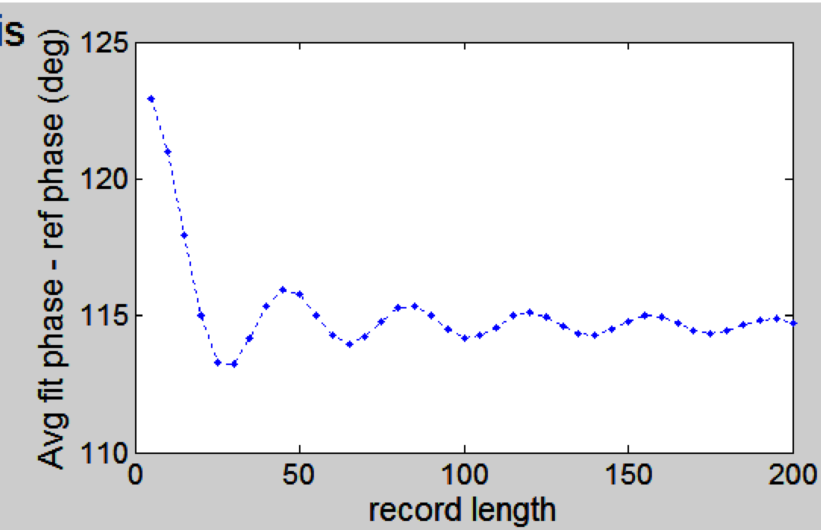
Use $x_k[N]$ to get amplitude and phase

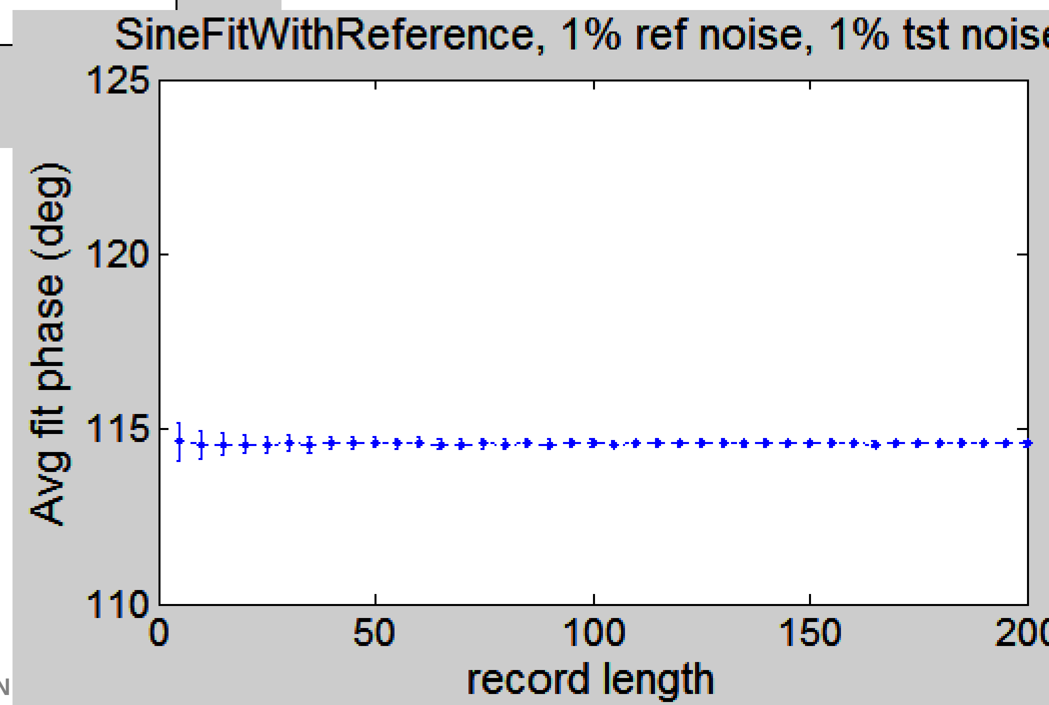
Constants depend on record length

Apply a correction to center 201.25 MHz in frequency bin k

Compare to Goertzel Algorithm







Frequency-domain approach has problems with short records

- Introduction: LANSCE and the BPPM system
- Initial algorithm and its shortcomings
- Improved algorithm
- Details of implementation
- Comparison to another algorithm
- **Signal processing modes**
- Performance
- Summary

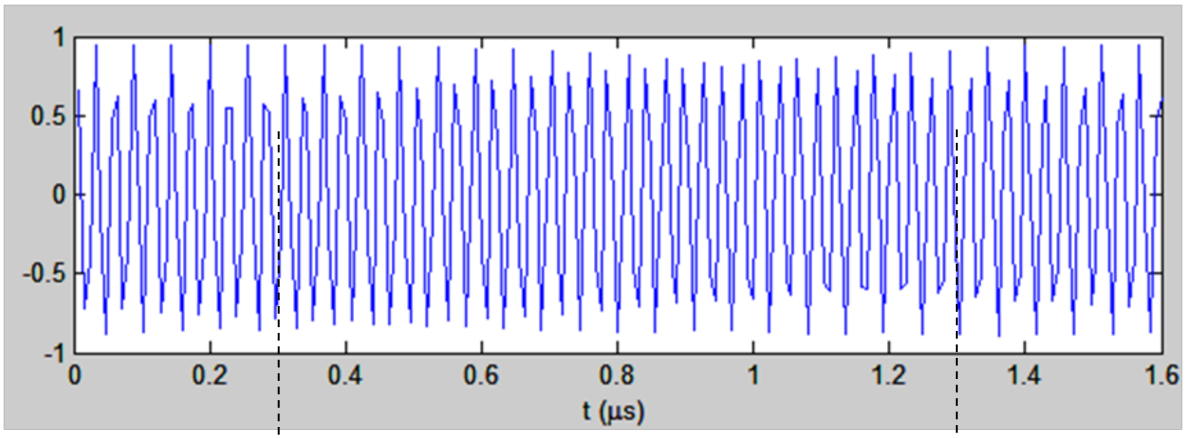
Sequencing logic

3 processing modes

Un-chopped	Minipulse	Single-micropulse
Continuous stream of micropulses	Bursts of micropulses	Ringing filter engaged
1 μs-long blocks (user-defined)	1 measurement per minipulse	Position only (no phase)

Sequencing logic: Unchopped mode

625μs
macropulse



Many measurements per macropulse

Previous block

Block for analysis
(1 μs typ.)

Next block

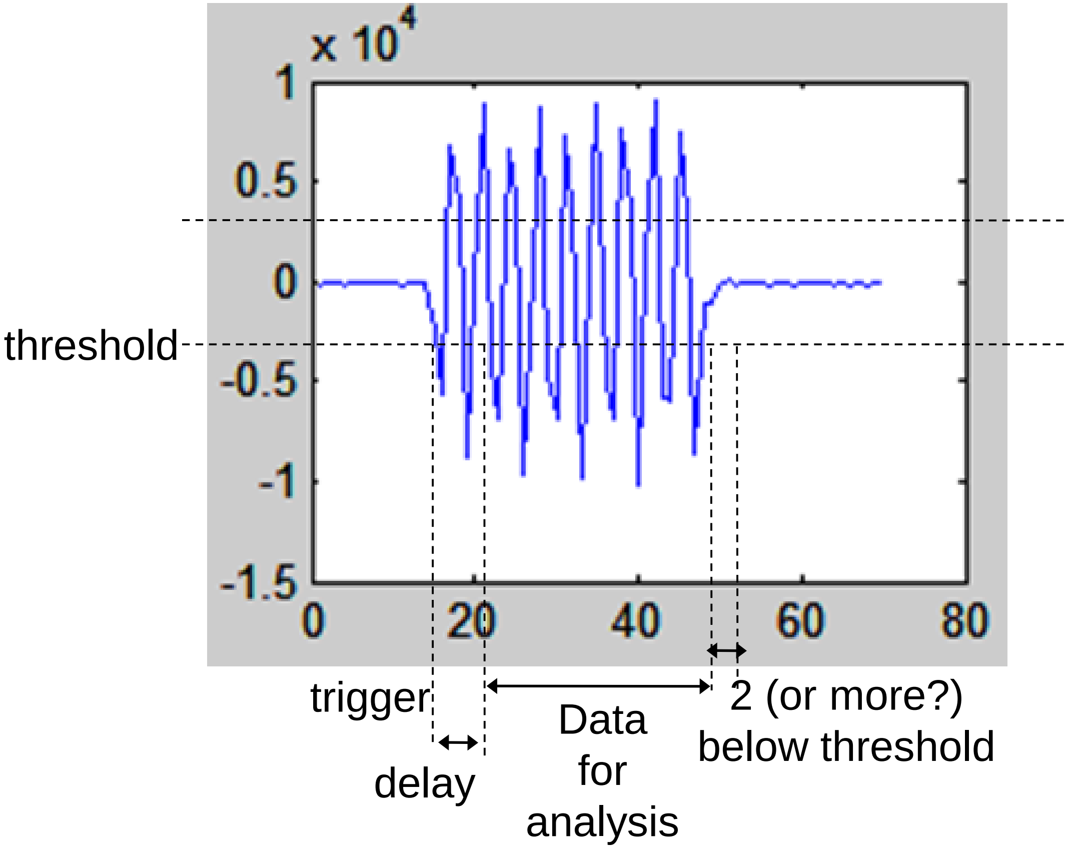
Sine fit
(I&Q demod)

Single measurement of
H, V, ϕ
for the block

Sequencing logic: Minipulse mode

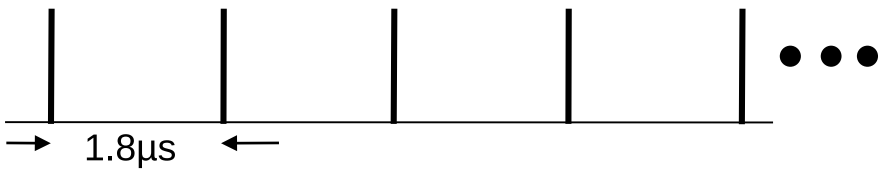
Self triggering

Rep rate and length of minpulses vary



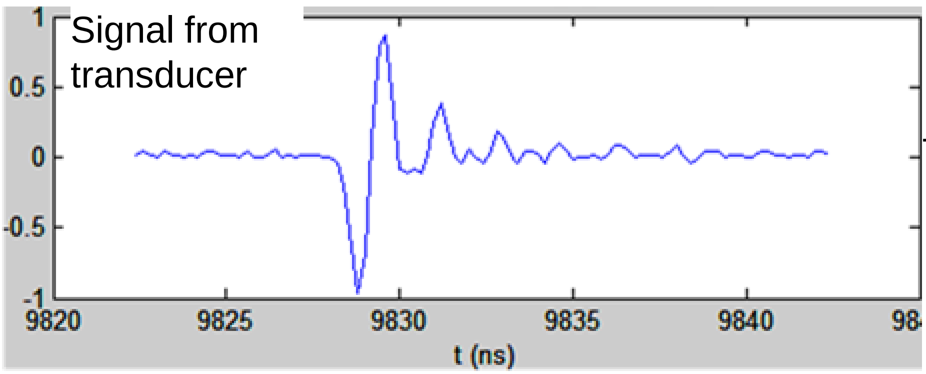
Sine fit (I&Q demod) → Single measurement of H, V, ϕ for the minipulse

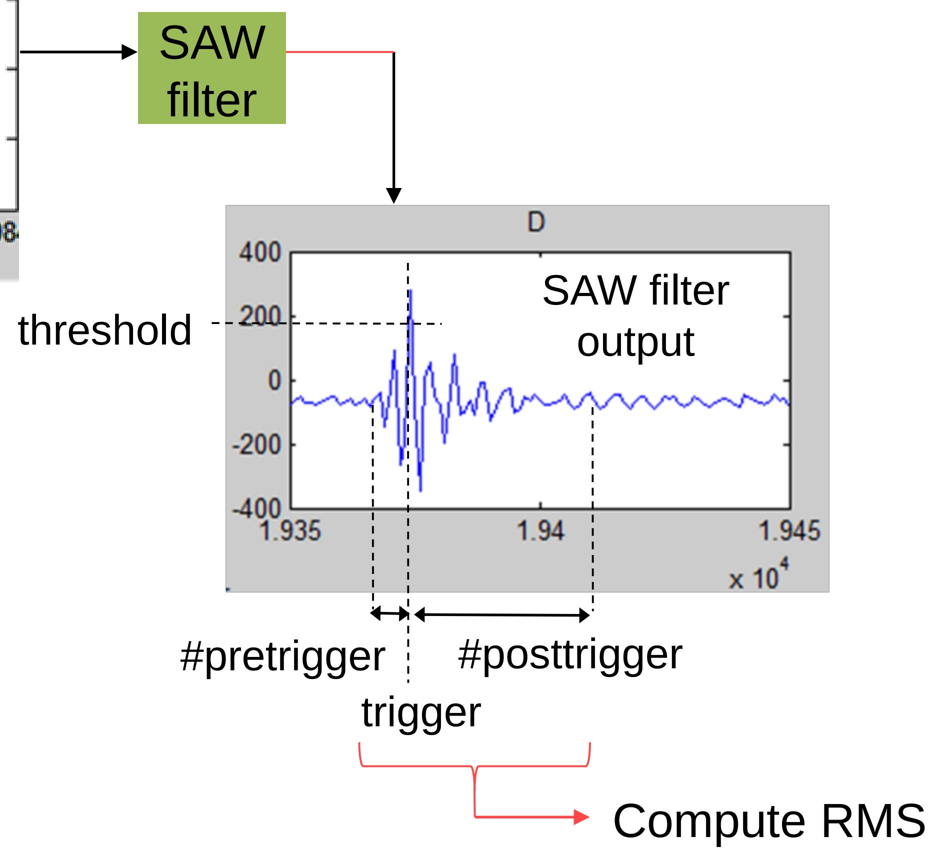
Sequencing logic: Single-micropulse mode



~350 single-micropulse beam pulses.

~2½× normal µpulse.





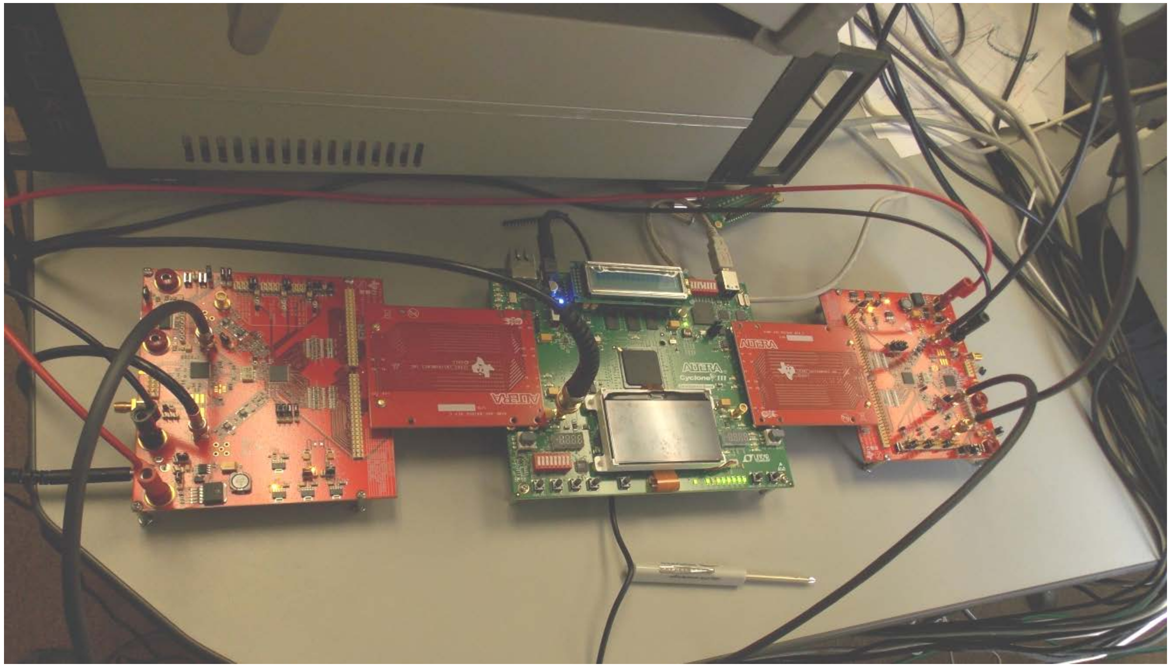
Single measurement of
H, V (not ϕ)
for the micropulse

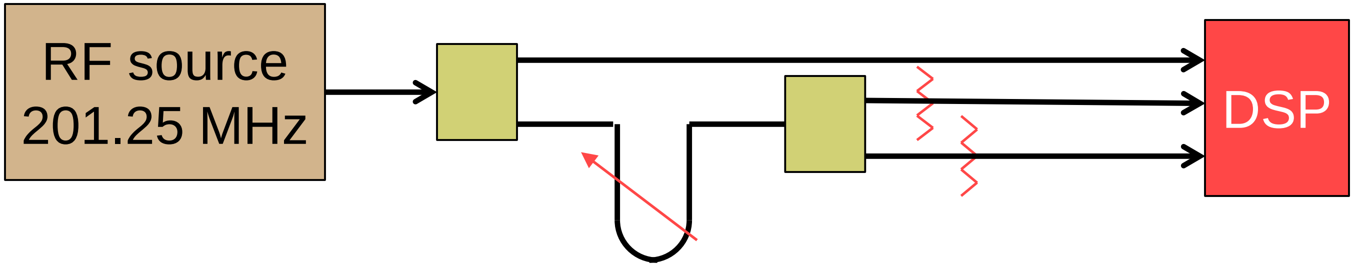
Performance

- Introduction: LANSCE and the BPPM system
- Initial algorithm and its shortcomings
- Improved algorithm
- Details of implementation
- Comparison to another algorithm
- Signal processing modes
- **Performance**
- Summary

Measurements: Bench tests of DSP algorithm

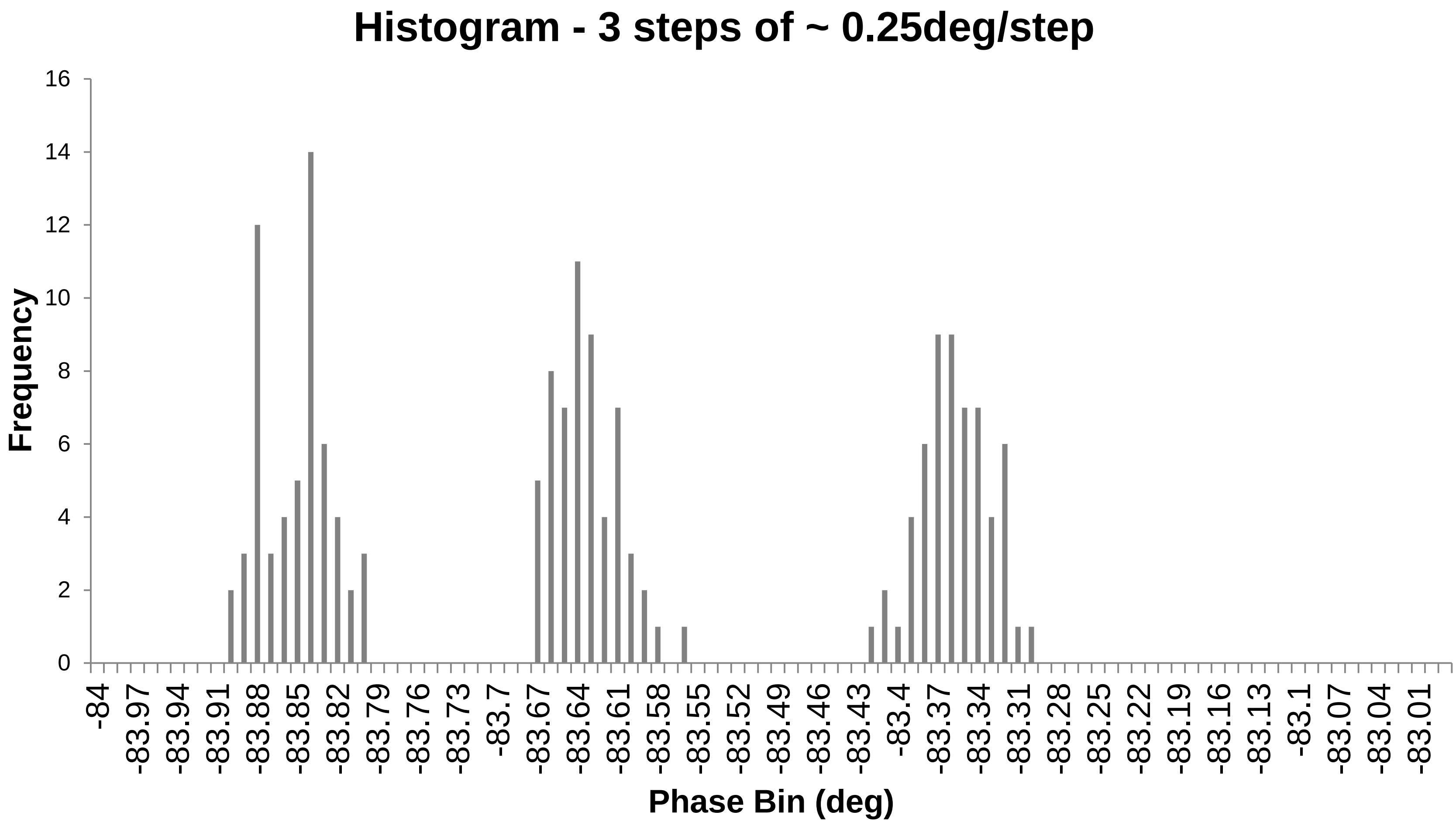
- Used very similar digitizers and FPGA
- 2 electrodes + reference
- RF synthesizer, attenuators, phase shifter, etc.



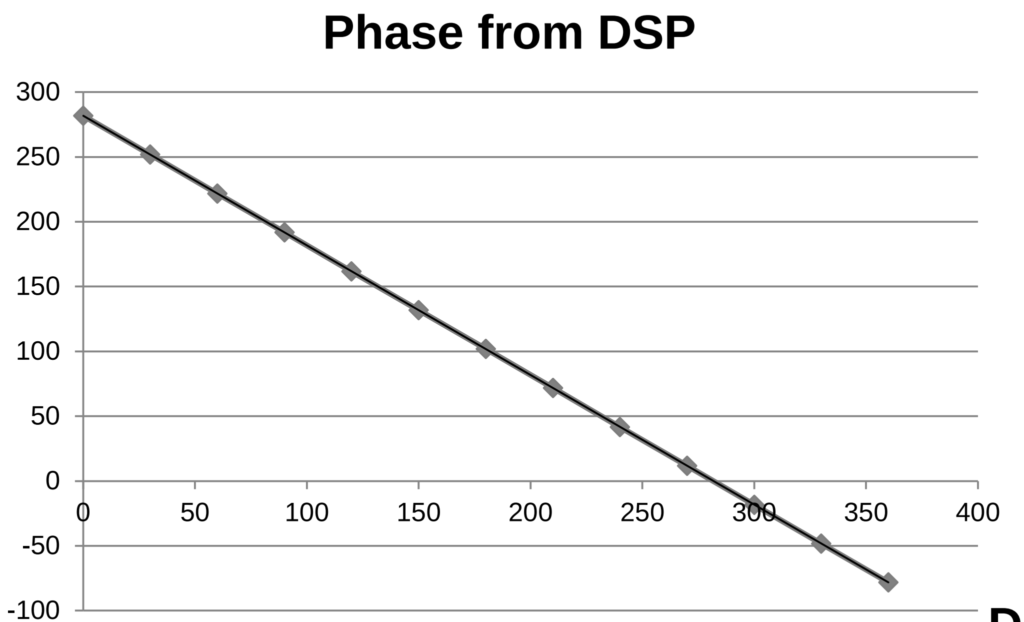


Performance: Phase resolution - bench test

0.25° is resolved well

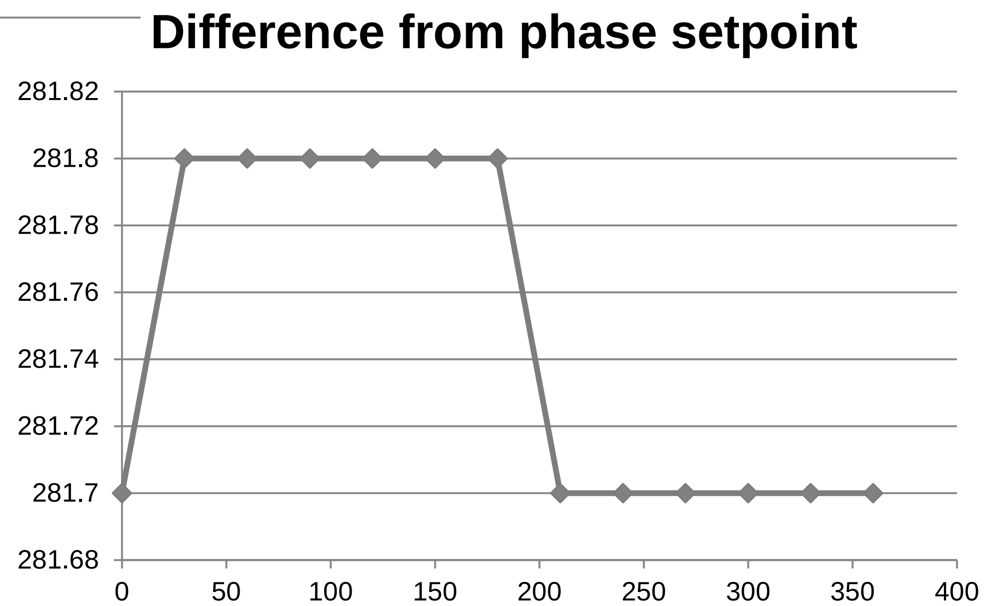


Performance: Bench test – Phase sweep



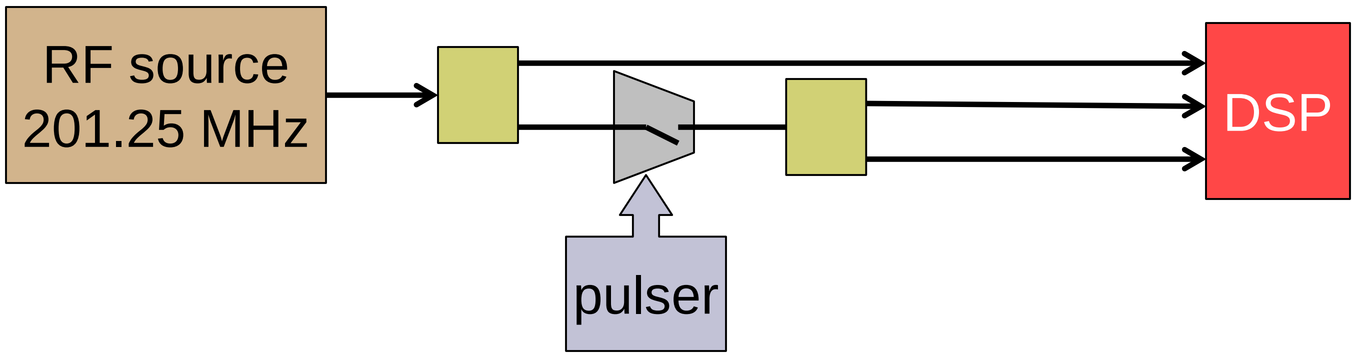
Signal level: ±200 counts (2.5% FS)

Analysis block: 100 samples



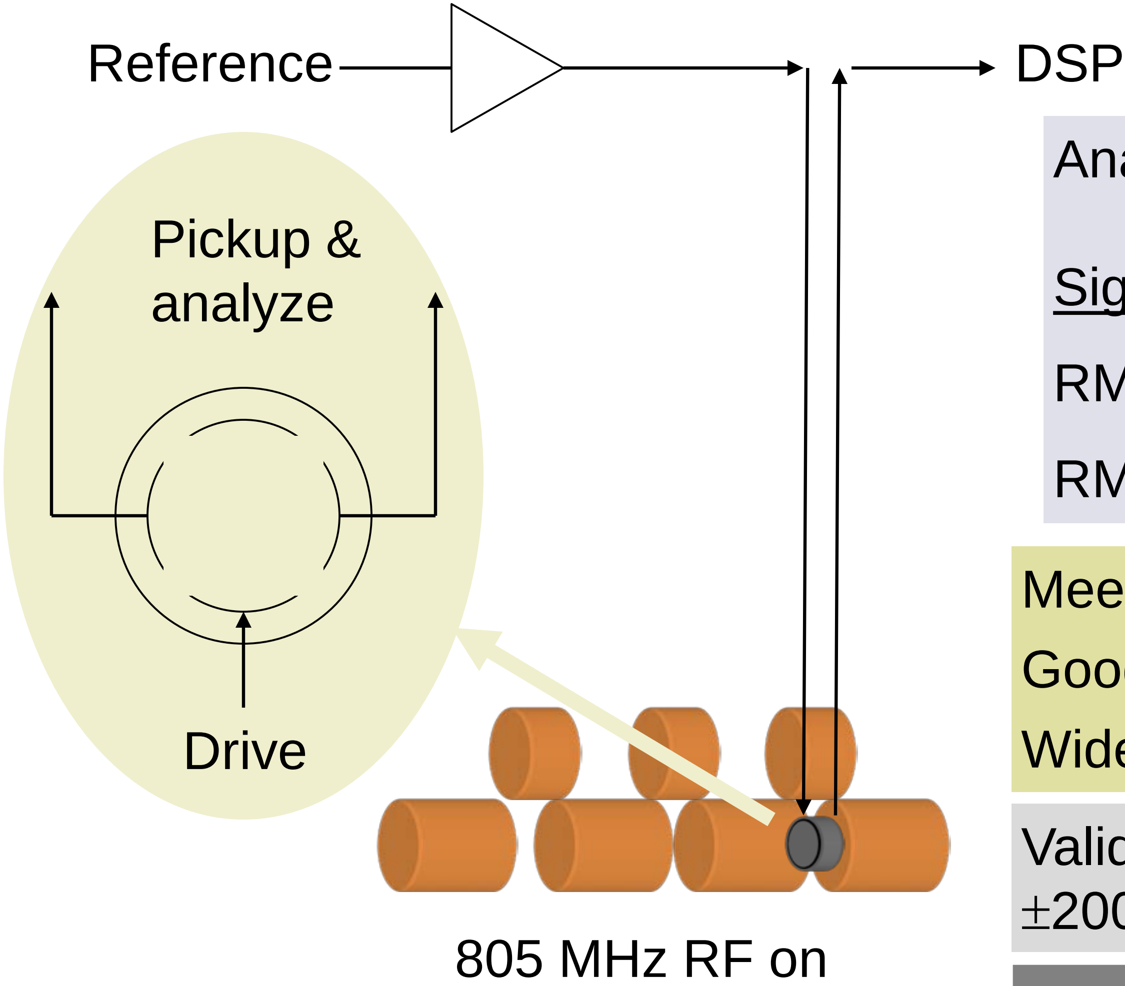
Phase accuracy specification is met

Performance: Bench test – short minipulses



- **150ns – long minipulses (~ 17 samples)**
- **Signal level: ±2000 counts (25% FS)**
- **ϕ rms = 0.7° (spec is 2 °)**
- **Position rms < 0.02 mm (spec is 0.5 mm)**

Performance: Beam-environment tests



Analysis block :100 samples

Signal amplitude ≈ 3% full scale

RMS phase: <0.06°

RMS position: <15μm

Meets the spec

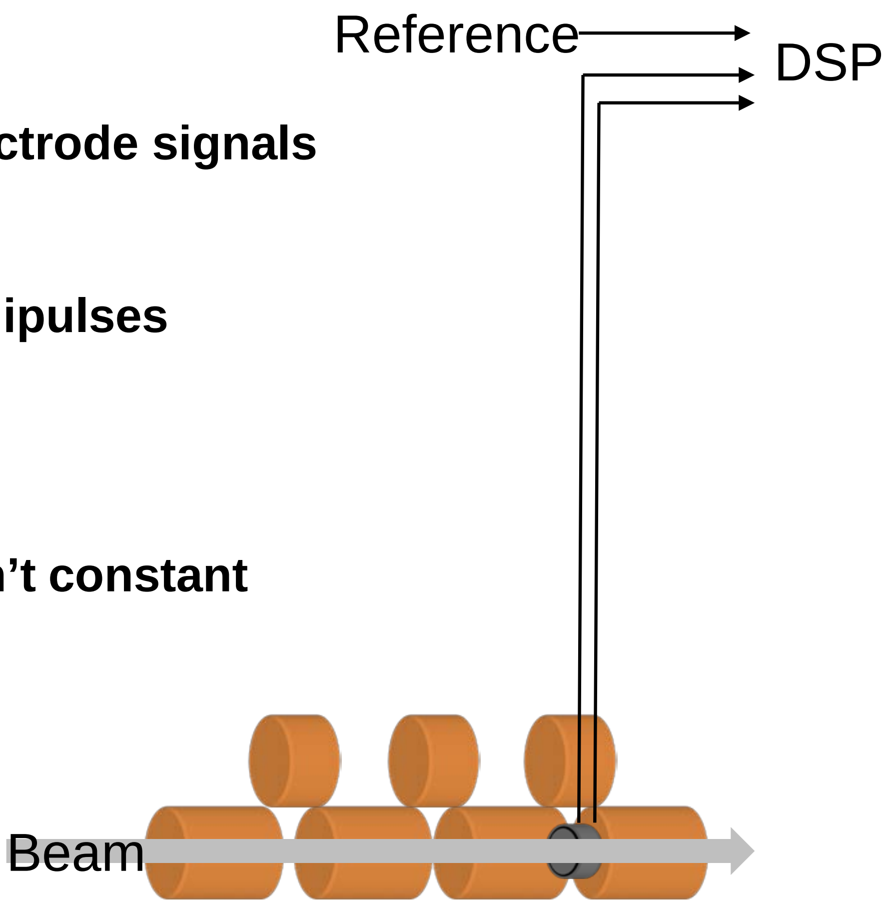
Good precision

Wide dynamic range (about 50dB)

Validates bench tests. For example:
±200 counts signal level (2.5% FS)

	Bench	Beamline
ϕ rms	0.060°	0.058°
Position rms	13 μm	12 μm

Performance: Beam measurements



Single low-pass coaxial filters on electrode signals

PSR beam with 1000 290ns-long minipulses

<0.25° RMS phase

0.28 mm RMS horizontal position

(the beam position probably wasn't constant during the measurement)

Summary

- An ad-hoc algorithm for determining beam position and phase works well
- Works with a wide range of pulse widths
- Is expensive in terms of FPGA resources
- Production systems are on order

Thanks for listening