First SPES School on Experimental Techniques with Radioactive Beams

INFN LNS Catania – November 2011

Experimental Challenges
Lecture 4: Digital Signal Processing

Tom Davinson

School of Physics & Astronomy

Objectives & Outline

Practical introduction to DSP concepts and techniques

Emphasis on nuclear physics applications

I intend to keep it simple …

… *even if it's not* …

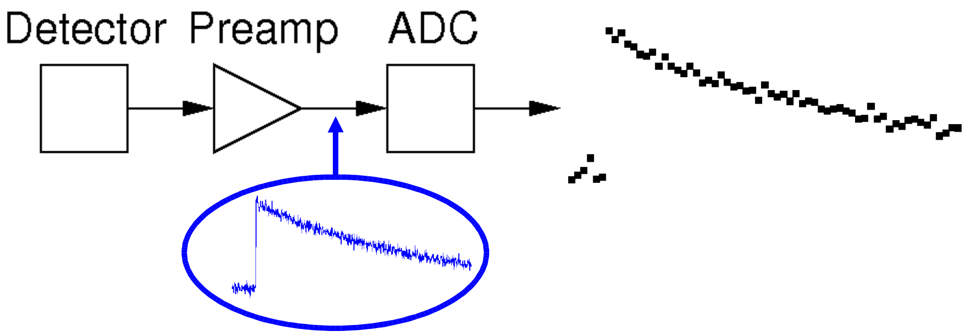
… I don't intend to teach you VHDL!

- Sampling Theorem
- Aliasing
- Filtering? Shaping? What's the difference? … and why do we do it?
- Digital signal processing
- Digital filters
  - semi-gaussian, moving window deconvolution
- Hardware
- To DSP or not to DSP?
- Summary
- Further reading

Sampling

Sampling

Periodic measurement of analogue input signal by ADC



Sampling Theorem

An analogue input signal limited to a bandwidth f_{BW} can be reproduced from its samples with no loss of information if it is regularly sampled at a frequency

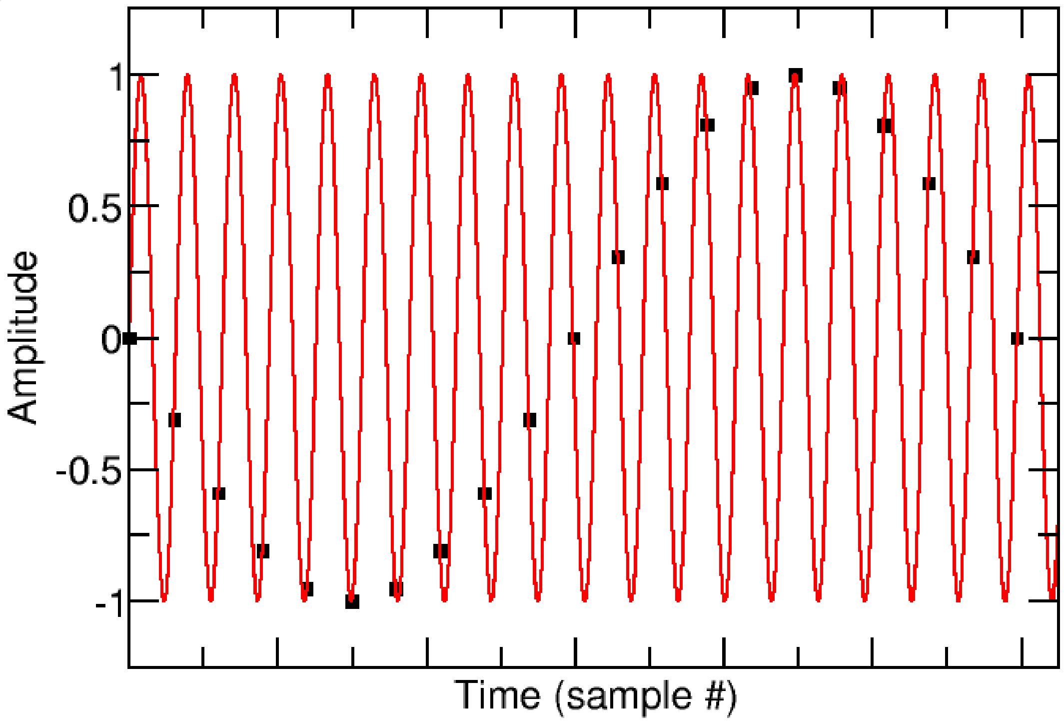
$$f_s \geq 2f_{BW}$$

The sampling frequency $f_s = 2f_{BW}$ is called the Nyquist frequency (rate)

Note: in practice the sampling frequency is usually >*5x* the signal bandwidth

Aliasing: the problem

Continuous, sinusoidal signal frequency f sampled at frequency f_s ($f_s < f$)

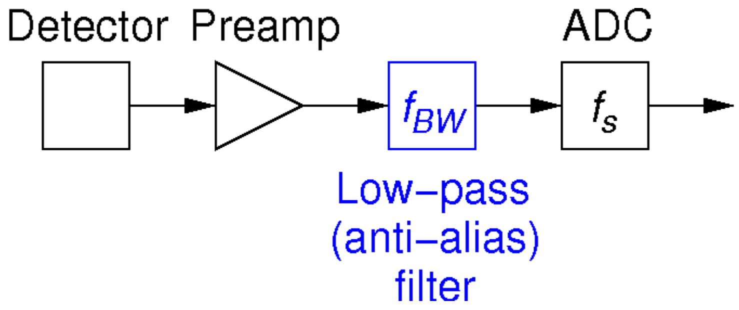


Aliasing misrepresents the frequency as a *lower* frequency $f < 0.5f_s$

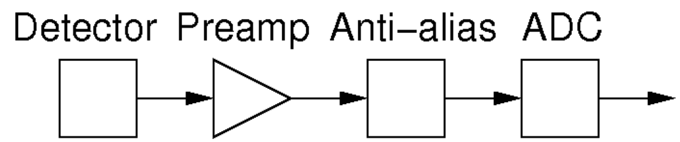
Aliasing: the solution

Use low-pass filter to restrict bandwidth of input signal to satisfy Nyquist criterion

$$f_s \geq 2f_{BW}$$



Digital Signal Processing



..what next?
Digital Signal Processing

Digital signal processing is the software controlled processing of sequential data derived from a digitised analogue signal.

Some of the advantages of digital signal processing are:

- functionality
  possible to implement functions which are difficult, impractical or impossible to achieve using hardware, e.g. FFT, ‘perfect’ filters etc.
- stability
  post-digitisation the data is immune to temperature changes, power supply drifts, interference etc.
- noiseless
  post-digitisation no additional noise from processing
  (assuming calculations are performed with sufficient precision)
- linearity
  … *perfect!*

Objectives of Pulse Shaping

Filter – modification of signal bandwidth *(frequency domain)*

Shaper – modification of signal shape *(time domain)*

Fourier transform – calculate frequency domain from time domain
(and vice versa)

Signal modified in frequency domain $\Rightarrow$ signal shape modified

Filter = Shaper

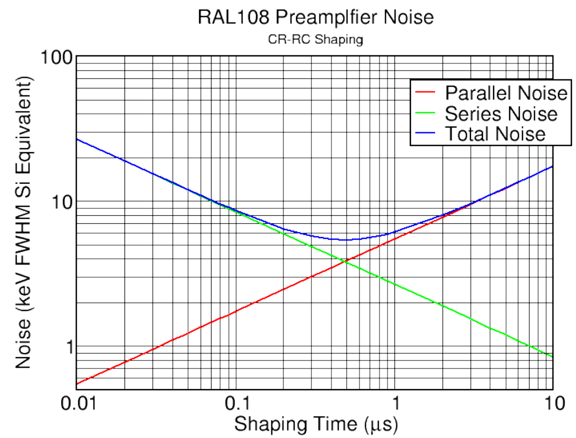
Why use pulse shaping?

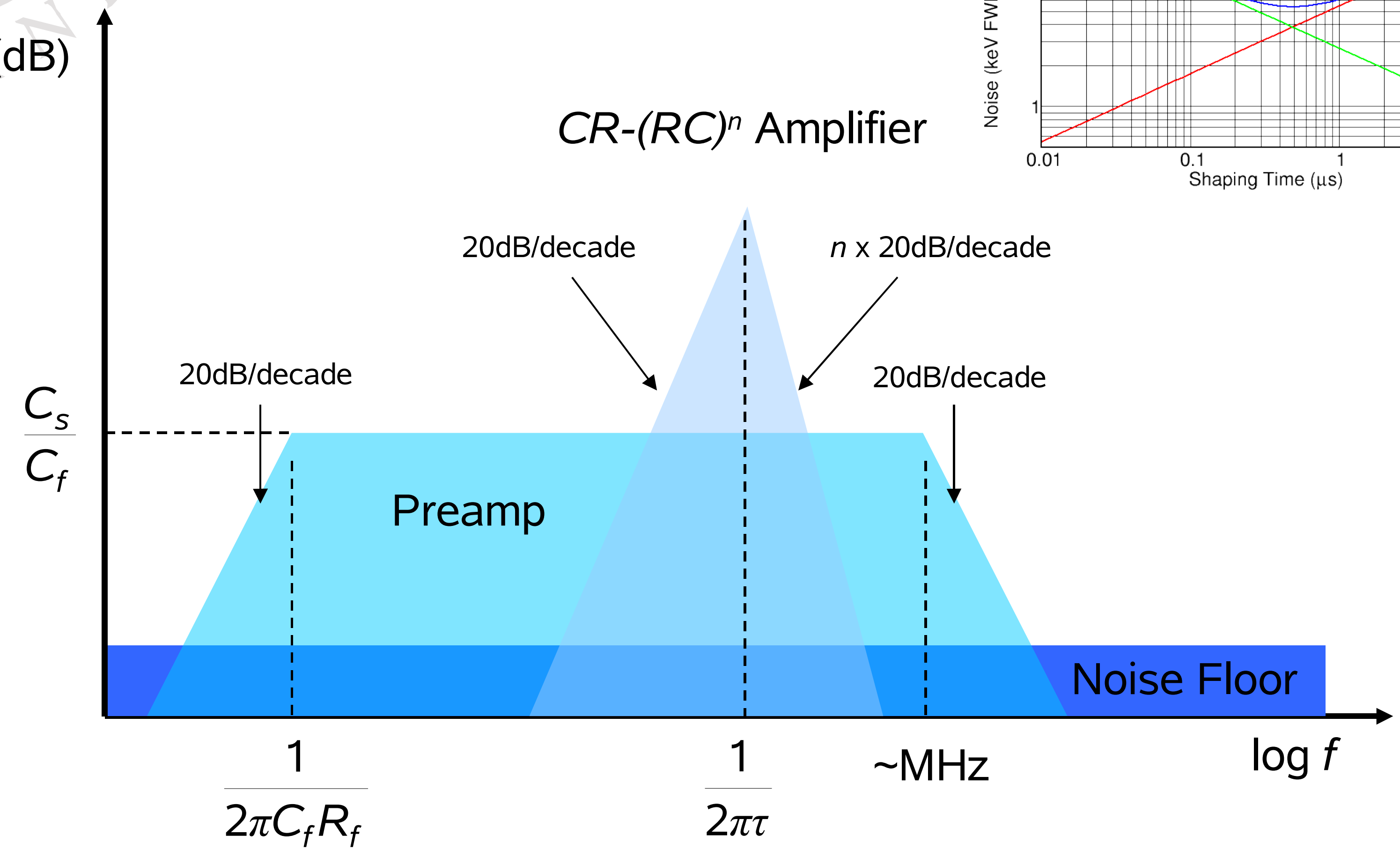
- Signal to noise ratio optimisation
- Throughput optimisation
- Ballistic deficit minimisation
- ADC input signal conditioning

- Conflicting requirements – compromise solutions

Signal to Noise Ratio Optimisation

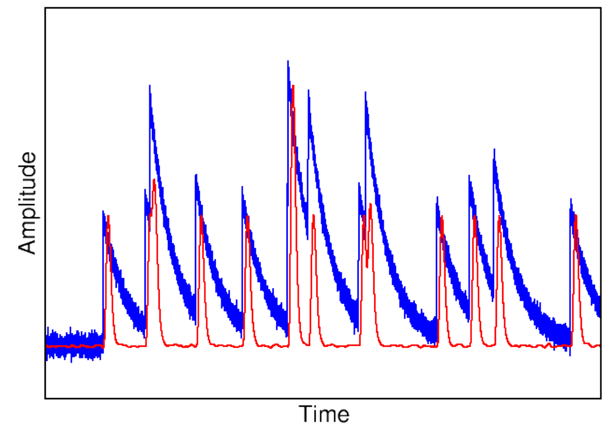
- Reduce bandwidth to optimise signal to noise ratio
- Optimum shaping time for minimum noise

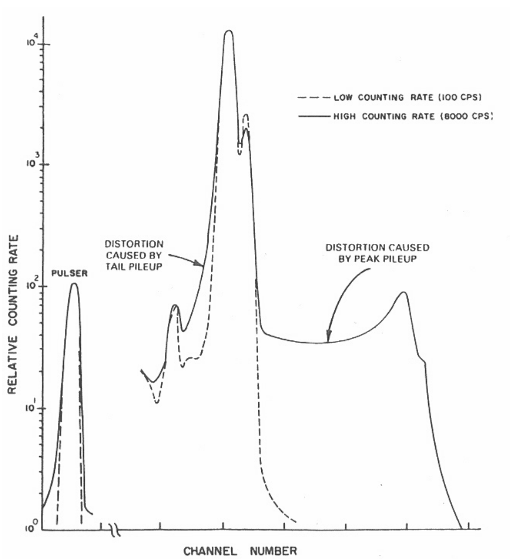




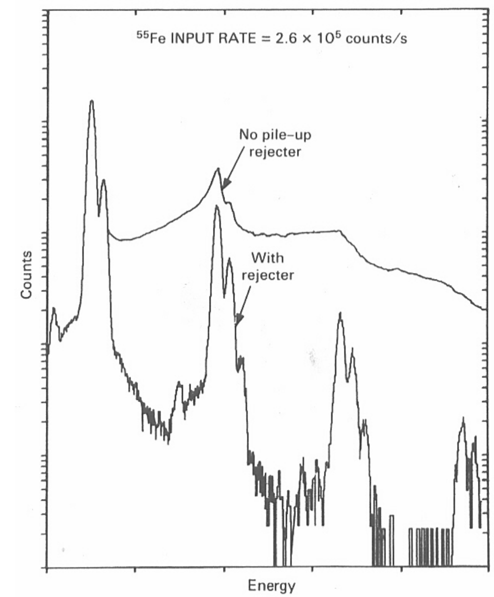
Throughput Optimisation

Minimise pulse width to minimise effects of pileup and maximise throughput
– short shaping times required





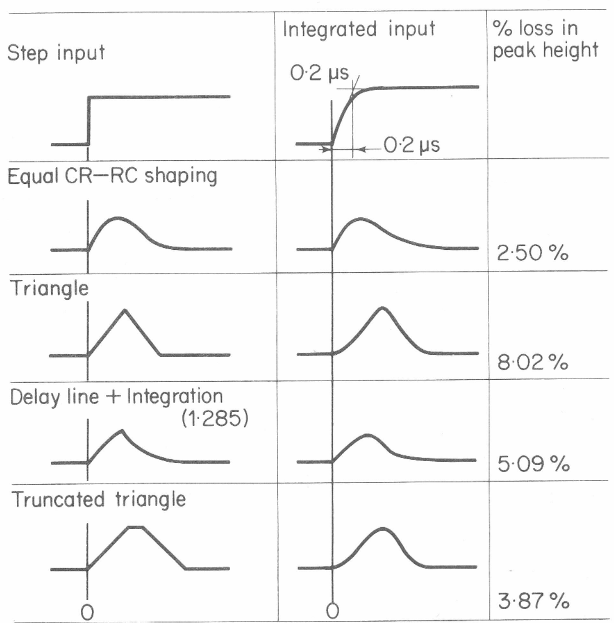
from L.Wielpowski & R.P.Gardner, NIM 133 (1976) 303

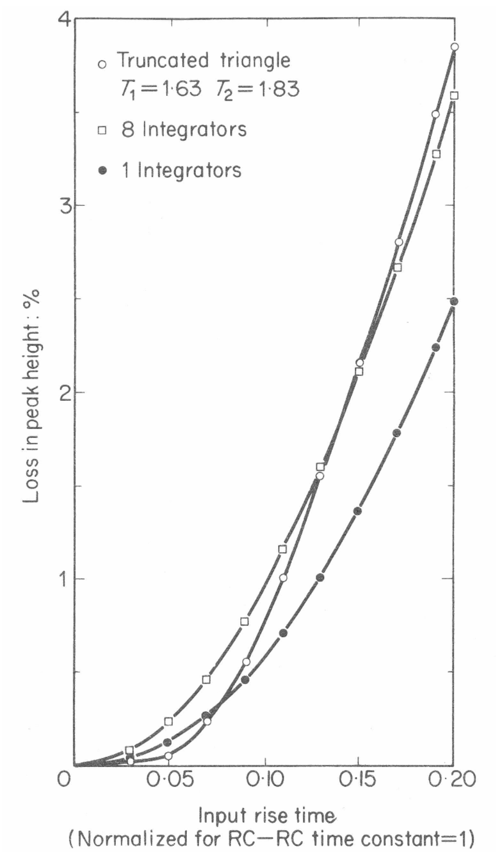


from F.S.Goulding & D.A.Landis, IEEE Trans. Nucl. Sci. 25 (1978) 896

Ballistic Deficit

from *Nuclear Electronics*, P.W. Nicholson, Wiley, 1974
K.Hatch, IEEE Trans. Nucl. Sci. NS15 (1968) 303





Require long shaping times compared to input risetime *variations*

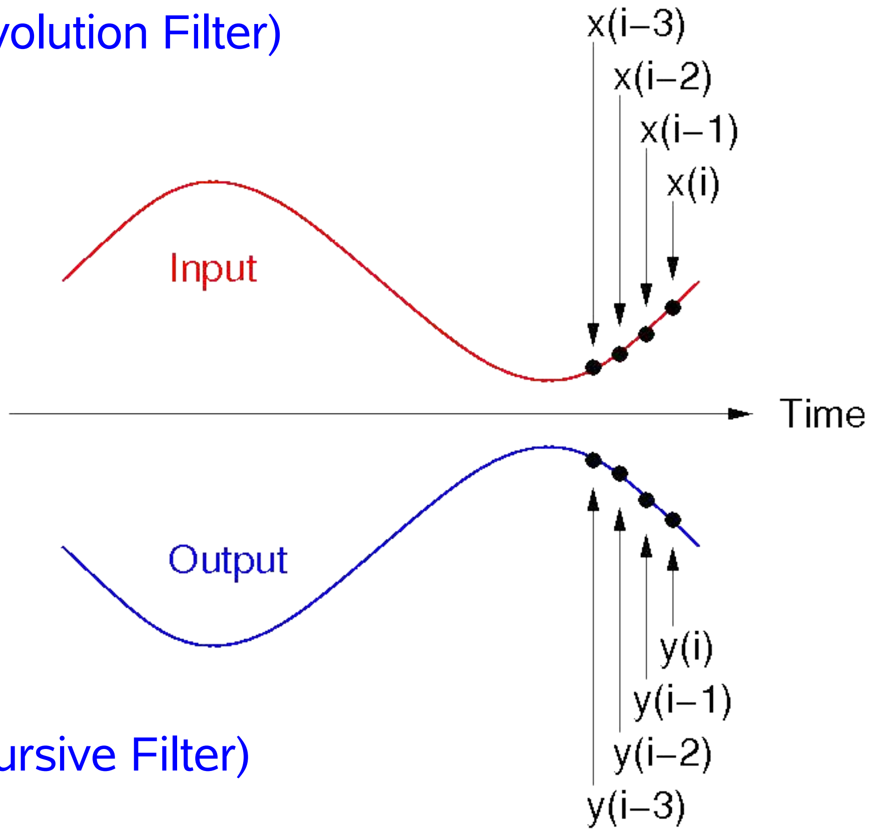
Digital Filters

Finite Impulse Response (FIR) filter (Convolution Filter)

$$y_i = a_0 x_i + a_1 x_{i-1} + a_2 x_{i-2} + \ldots$$

where a_0, a_1 … etc. are coefficients,
x_i, x_{i-1} … etc. the input data and
y_i, y_{i-1} … etc. the output data.

Example – moving average filter



Infinite Impulse Response (IIR) filter (Recursive Filter)

$$y_i = a_0 x_i + a_1 x_{i-1} + a_2 x_{i-2} + \ldots + b_1 y_{i-1} + b_2 y_{i-2} + b_3 y_{i-3} + \ldots$$

where b_1, b_2 … etc. are coefficients.

Example – simple first order (single pole) filter

Simple Recursive Filters

Low pass filter (integrator)

$$a_0 = 1 - z$$

$$b_1 = z$$

$$y_i = a_0 x_i + b_1 y_{i-1}$$

where $0 < z \leq 1$

High pass filter (differentiator)

$$a_0 = \frac{1+z}{2}$$

$$a_1 = -\frac{1+z}{2}$$

$$b_1 = z$$

$$y_i = a_0 x_i + a_1 x_{i-1} + b_1 y_{i-1}$$

For an analogue *RC* circuit, the time constant *RC* is the time to decay to 36.8% of the initial value: *d* is the number of samples it takes for a digital recursive filter to decay to the same level.

$$z = \exp\left(-\frac{1}{d}\right)$$

DSP Program: Semi-Gaussian Filter

```fortran
      PROGRAM semigauss

C     Number of samples = n
      INTEGER n
      PARAMETER (n = 1000000)

C     Number of poles = n_poles
      INTEGER n_poles
      PARAMETER (n_poles = 6)

C     Gain=1/(n_poles**n_poles
C          *exp(-n_poles)/n_poles!)
      REAL gain
      PARAMETER (gain = 6.22575 )

C     Time constant = tc samples
      REAL tc
      PARAMETER (tc = 20.0)

C     Pole zero correction = pz samples
      REAL pz
      PARAMETER (pz = 500.0)

      INTEGER i, j
      REAL a0, a1, b0
      REAL x(0:n-1), y(0:n-1), z(0:n-1)
      REAL t(0:n-1)

C     Read input data
      DO i = 0, n - 1
       READ( 5, * ) t( i ), x( i )
      ENDDO

C     Single pole high pass filter
C     with pole-zero correction
      b1 = EXP( -1.0 / tc )
      a0 = (1.0 + b1) / 2.0
      a1 = - (1.0 + b1) / 2.0

      DO i = 1, n - 1
       y( i ) = b1 * y( i - 1 ) + a0 * x( i )
     +          + a1 * x( i - 1 ) + x( i - 1 ) / pz
      ENDDO

C     n-pole low pass filter
      b1 = EXP( -1.0 / tc )
      a0 = 1.0 - b1

      DO j = 1, n_poles
       DO i = 1, n - 1
        z( i ) = b1 * z( i - 1 ) + a0 * y( i )
       ENDDO
       DO i = 1, n - 1
        y( i ) = z( i )
       ENDDO
      ENDDO

C     Write semi-gaussian filter output
      DO i = 1, n - 1
       WRITE( 6, * ) t( i ), gain * z( i )
      ENDDO

      STOP
      END
```

FORTRAN77 source code

Input Data

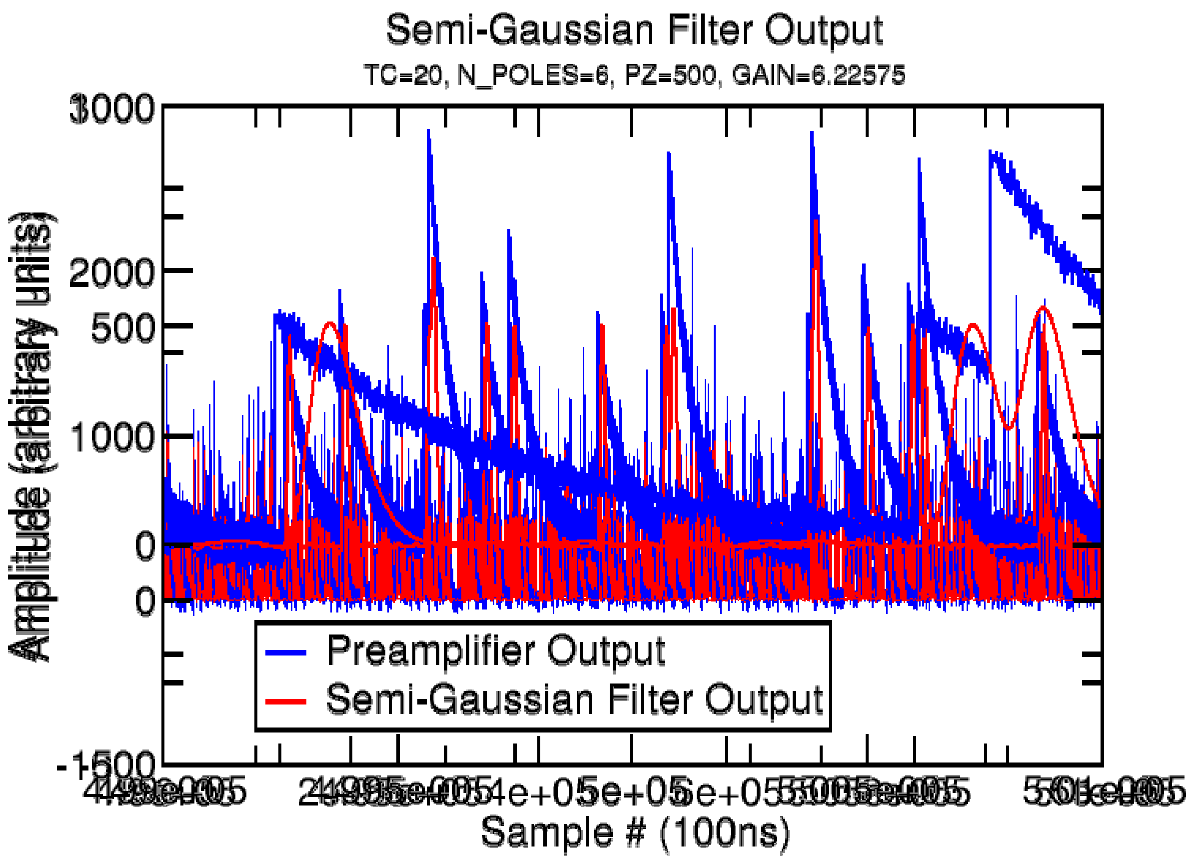
• Samples	1M
• Events	1000
• Preamplifier output amplitude	500
• Preamplifier decay time τ	500 samples
• Signal to noise ratio	25:1

Assuming we sample at 10MHz (100ns per sample)

• Sample history	0.1s
• Mean event rate	10kHz

The same input dataset will be used for all of the examples

Semi-Gaussian Filter Output



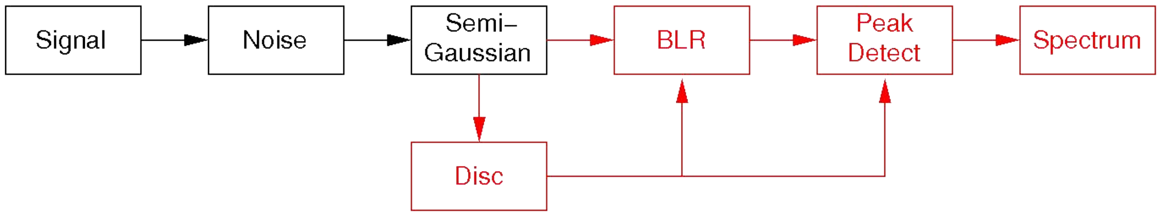
Semi–Gaussian Filter contd.

In practice, the semi-gaussian filter algorithm described would be elaborated to include one, or more, of the following:

- Constant fraction (or other) discrimination
- Pile-up rejection (PUR)
- Baseline restoration (BLR)
- Ballistic deficit correction
- Peak detection
- Pulse shape analysis (PSA)
- Integral and differential non-linearity correction (INL/DNL)

Example

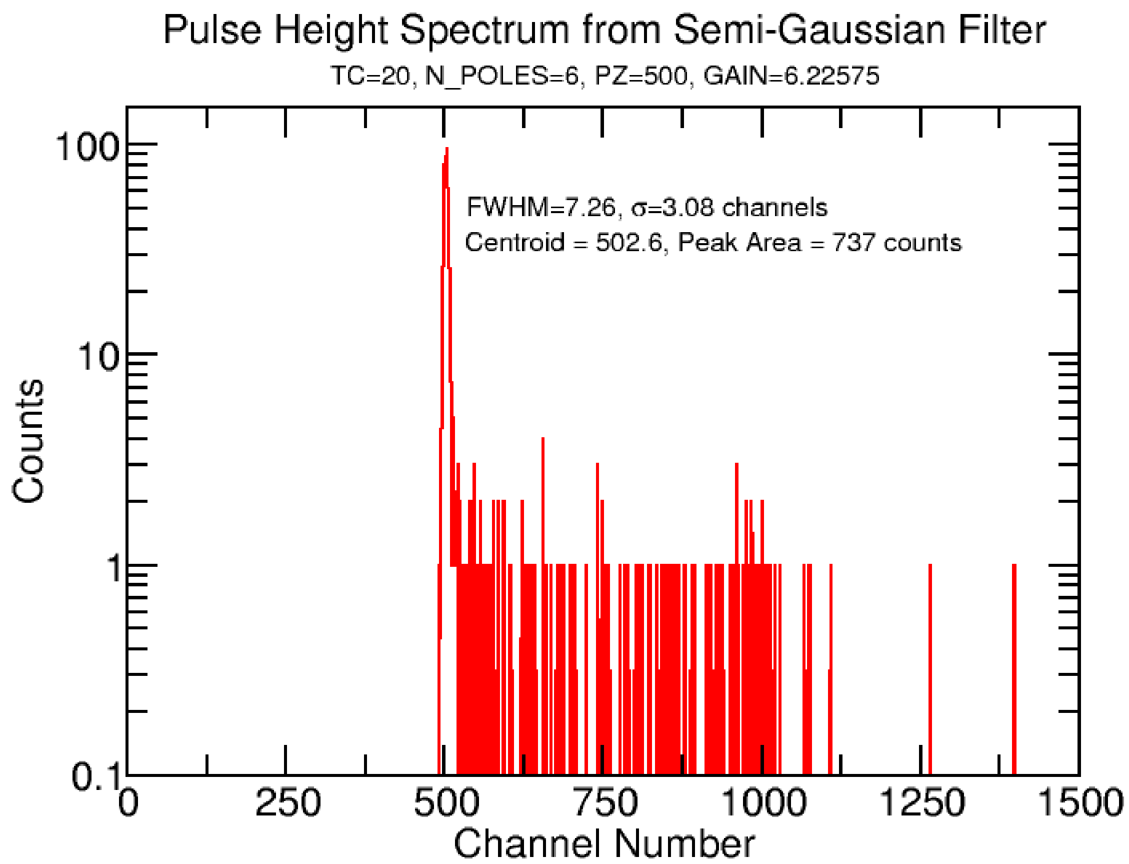
Add simple discriminator and implement baseline restorer and peak detect algorithms to generate pulse height spectra



Pulse Height Spectrum from Semi-Gaussian Filter

Preamplifier signal/noise ratio	25:1
Semi-gaussian signal/noise ratio	160:1

Input rate 10kHz, ~74% events no pile up



Moving Window Deconvolution (MWD)

MWD filter commonly used for X-ray and γ-ray detectors

For a preamplifier with decay time τ

Deconvolved Signal Differentiation Moving Average

$$MWD_M(i) = x(i) - x(i-M) + \frac{1}{\tau}\sum_{j=i-M}^{i-1} x(j)$$

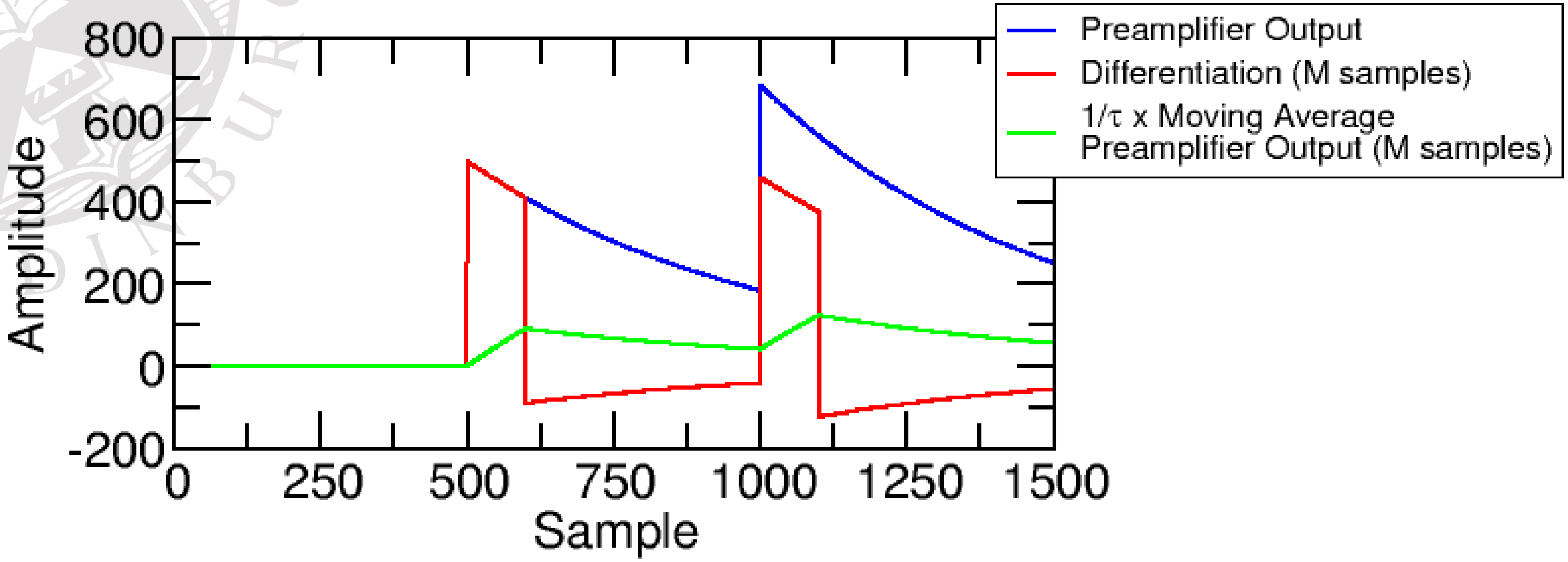
$$T_M^L(i) = \frac{1}{L}\sum_{j=i-L}^{i-1} MWD_M(j)$$

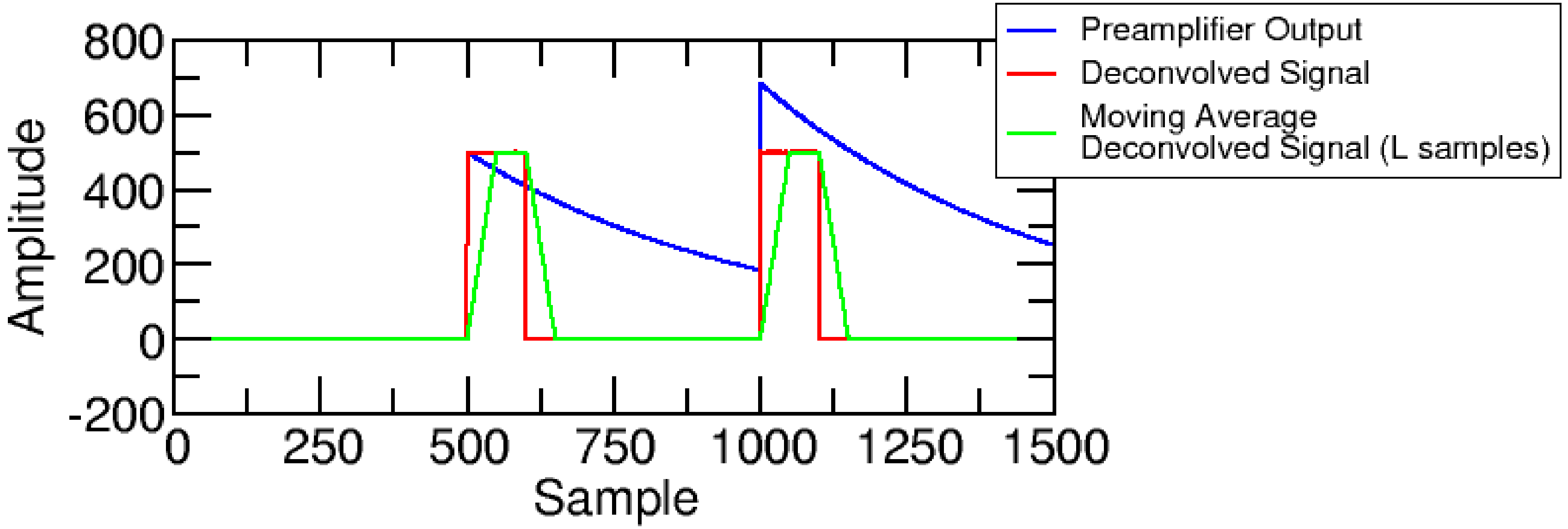
Moving Average (low pass filter)

Trapezoidal shaping: *L*<*M*, length of flat top *M-L*

Triangular shaping: *L*=*M*

Moving Window Deconvolution (MWD) contd.



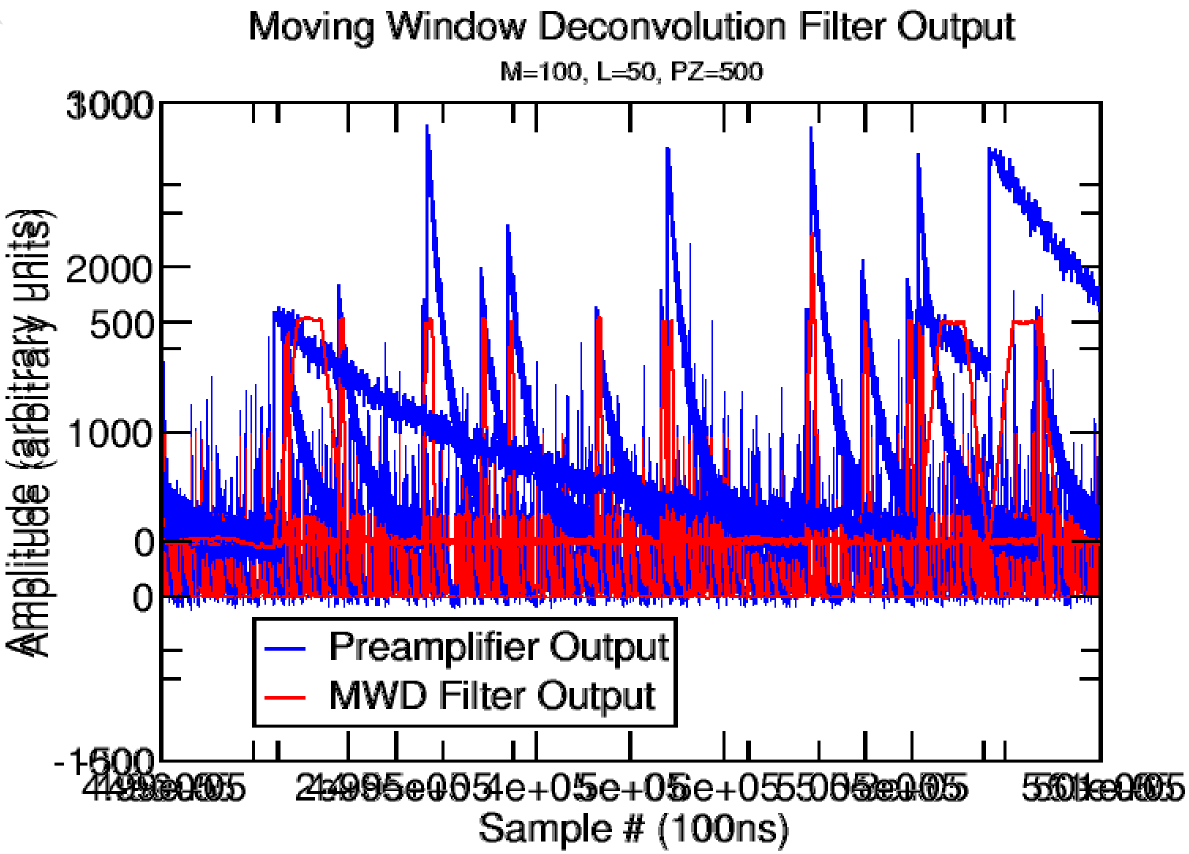


DSP Program: MWD

```
      PROGRAM mwd

C     Number of samples = n
      INTEGER n
      PARAMETER (n = 1000000)
C     Deconvolution window = m samples
      INTEGER m
      PARAMETER (m = 100)
C     Moving average window = l samples
      INTEGER l
      PARAMETER (l = 50)
C     Pole zero correction = pz samples
      REAL pz
      PARAMETER (pz = 500.0)

      INTEGER i, j
      REAL d_m, ma_l(0:n-1)
      REAL ma_m, mwd_m(0:n-1)
      REAL t(0:n-1), x(0:n-1)

C     Read input data
      DO i = 0, n - 1
       READ( 5, * ) t( i ), x( i )
      ENDDO

C     Moving window deconvolution
      DO i = m, n - 1
       d_m = x( i ) - x( i - m )
       ma_m = 0.0
       DO j = i - m, i - 1
        ma_m = ma_m + x( j )
       ENDDO
       mwd_m( i ) = d_m + ma_m / pz
      ENDDO

C     Moving average
      DO i = l, n - 1
       ma_l( i ) = 0.0
       DO j = i - l, i - 1
        ma_l( i ) = ma_l( i ) + mwd_m( j )
       ENDDO
       ma_l( i ) = ma_l( i ) / l
      ENDDO

C     Write MWD filter output
      DO i = m, n - 1
       WRITE( 6, * ) t( i ), ma_l( i )
      ENDDO

      STOP
      END
```

FORTRAN77 source code

Moving Window Deconvolution Filter Output

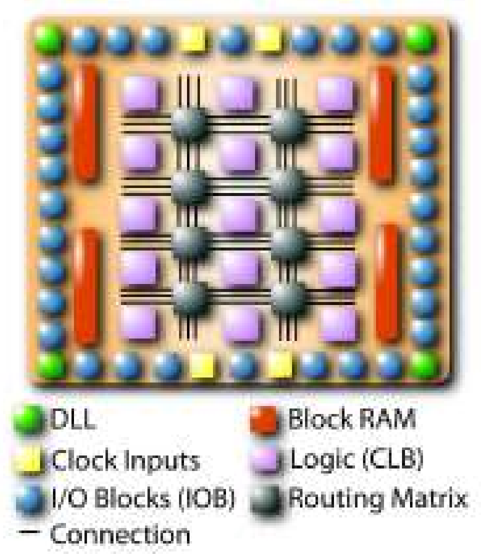


Digital Signal Processor

Specific hardware to implement the software controlled processing of sequential digital data derived from a digitised analogue signal.

• DSP algorithms usually implemented with programmable logic devices e.g. Field Programmable Gate Arrays (FPGAs) from Xilinx, Altera etc.

• FPGA consists of *lots (and lots)* of
    configurable logic blocks (CLBs)
    configurable interconnections
    configurable I/O blocks (IOBs)
    RAM
    etc.



from M.Lauer, PhD thesis, 2004

• FPGAs are *very* powerful devices

Digital Signal Processor contd.

- Design by high level abstractions with hardware description languages (HDLs) e.g. VHDL, Verilog

VHDL code extract – signed 16-bit adder *courtesy* Ian Lazarus, CCLRC DL

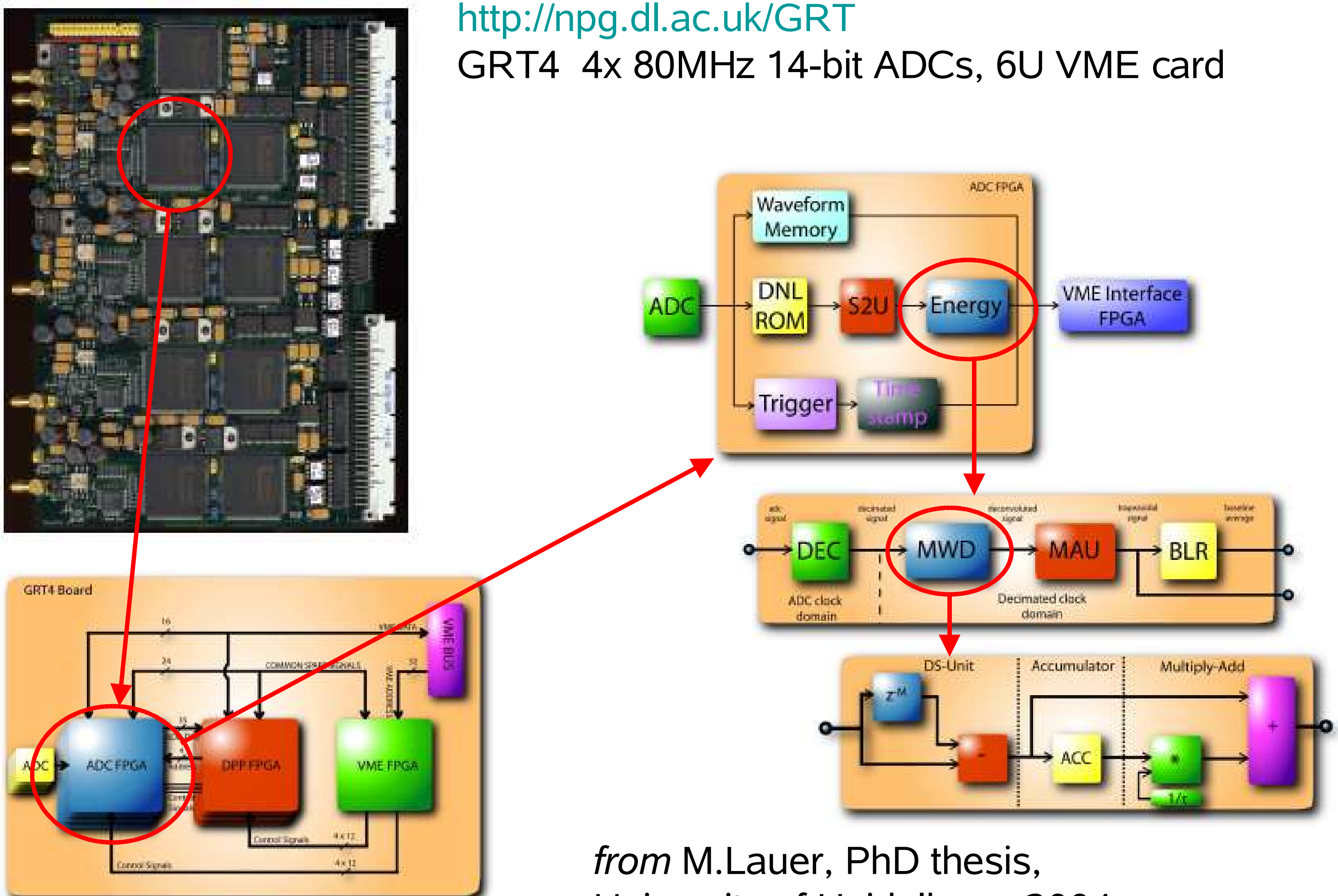
```
architecture Behavioral of add_signed is

SIGNAL temp: std_logic_vector(width downto 0);
SIGNAL ta: std_logic_vector(width downto 0);
SIGNAL tb: std_logic_vector(width downto 0);

begin

temp <= ta + tb + ("0"&CarryIn);

process(signed, A, B, CarryIn)
begin
        -- signed input
        case signed is
                when '1' => ta(width-1 downto 0) <=        A;
                                                           ta(width) <= A(width-1);
                                                           tb(width-1 downto 0) <=B;
                                                           tb(width) <= B(width-1);

        -- unsigned input
                when others => ta(width-1 downto 0) <=     A;
                                                           ta(width) <= '0';
                                                           tb(width-1 downto 0) <=B;
                                                           tb(width) <= '0';

        end case;
end process;
```

- HDL code is used to
  - simulate, optimise and synthesise design
  - generate data required to configure FPGA
- Result – customised, high performance computer
- Near real-time performance

Digital Signal Processor: GRT4

http://npg.dl.ac.uk/GRT

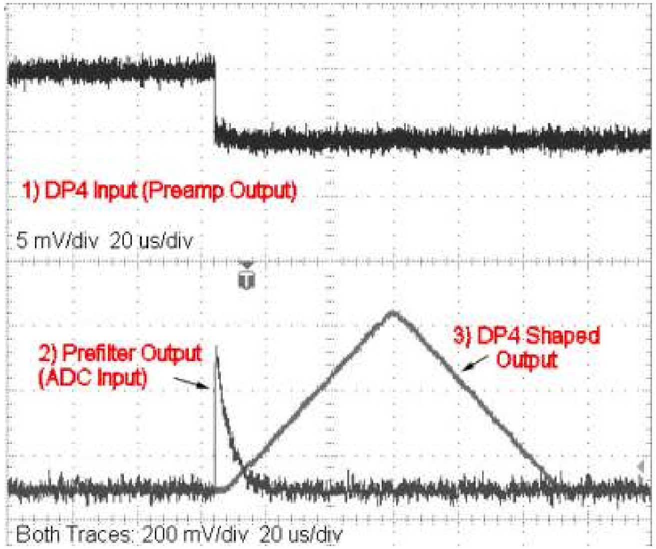
GRT4 4x 80MHz 14-bit ADCs, 6U VME card

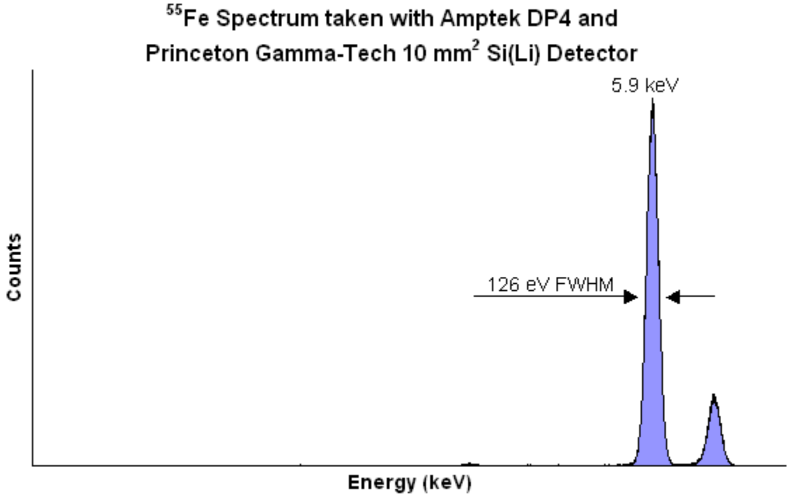


from M.Lauer, PhD thesis,
University of Heidelberg, 2004

Digital Signal Processor: Amptek DP4

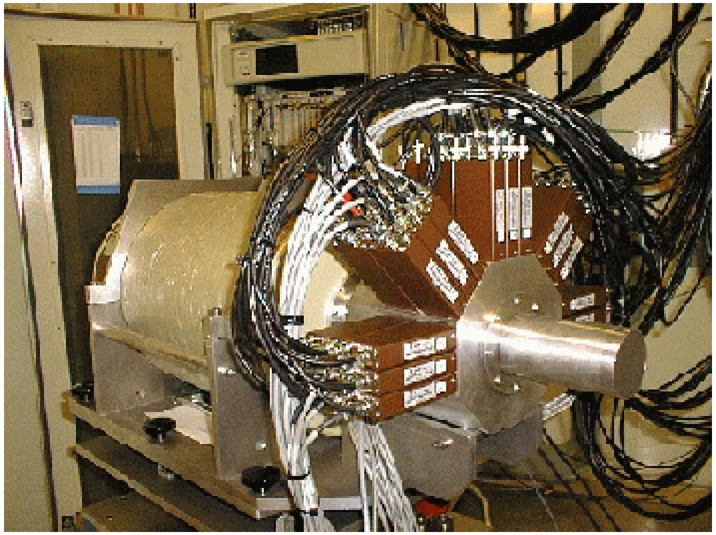
Size: 8.9cm x 6.4cm





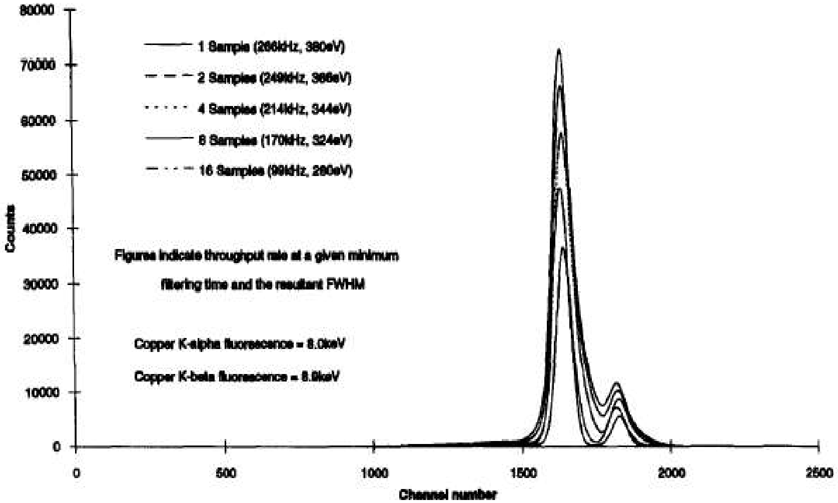
http://www.amptek.com/dpp.html

Digital Signal Processor: XSPRESS



- *Classic DSP application*
- EG&G Ortec 30 x 30mm^2 HPGe EXAFS array
- CCLRC DL VME-based DSP instrumentation
- *Adaptive* digital signal processing algorithm
  filter bandwidth varied depending on time
  available to next event
- Throughput c. 400kHz/channel at 5% resolution
  (400eV FWHM Cu K_{α})

R.Farrow *et al*., NIM B97 (1995) 567



Digital Signal Processor: other examples

XIA DGF4

Miniball @ REX-ISOLDE

see Thorsten Kroll's lectures

DSSSDs + … @ HRIBF/ORNL

M.Karny *et al*. Phys. Rev. Lett. 90 (2003) 012502

http://fribusers.org/4_GATHERINGS/2_SCHOOLS/2010/talks/Grzywacz_1.pdf

CAEN V17xx Modules

see T.Marchi's presentation

AGATA

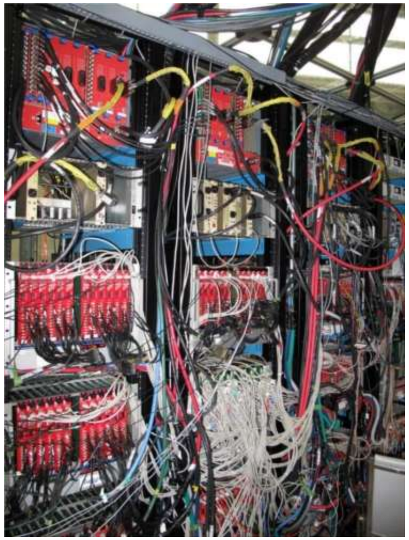
see E.Farnea's presentation

F.Recchia *et al*., NIM A 604 (2009) 555

The New World

Multichannel 100MSPS, 14-bit ADC modules

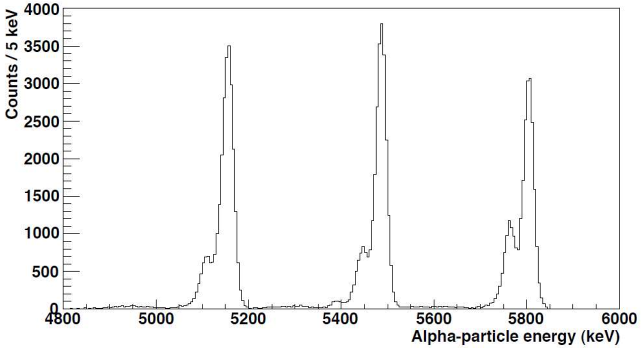
GRETINA + …	*http://grfs1.lbl.gov/*
GAMMASPHERE refit	
FMA 160x160 DSSSD	being upgraded
TIGRESS/SHARC	J.P.Martin *et al*., IEEE NS55 (2008) 84



TIGRESS VXI racks

32-fold segmented HPGe detectors

SHARC DSSSDs



C.A.Diget *et al.*, J. Inst. 6 (2011) P02005

To DSP or not to DSP?

Use DSP for …

resolution & throughput optimisation
variable detector pulse shapes

Use analogue signal processing for …

fast shaping
systems not sensitive to, or with fixed, detector pulse shapes
high density (low area, low power) applications

Expect …

ADCs with higher precision, speed & density
lower power & cost

more powerful FPGAs
an expanding range of applications

Summary

- DSP concepts are straightforward
  - you don’t need to be a rocket scientist to understand them

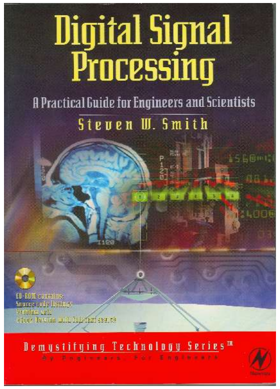
- *Real world* DSP implementations use FPGAs
  - this *is* rocket science
  - highly abstracted hardware design description
  - optimised generic design building blocks available
  - development, test and optimisation tools available
  - real time performance

- Other nuclear physics applications
  - spectrum analysis

- Wider applications
  - sound/image/video
  - neural networks
  - data compression
  - FFT
  - etc.

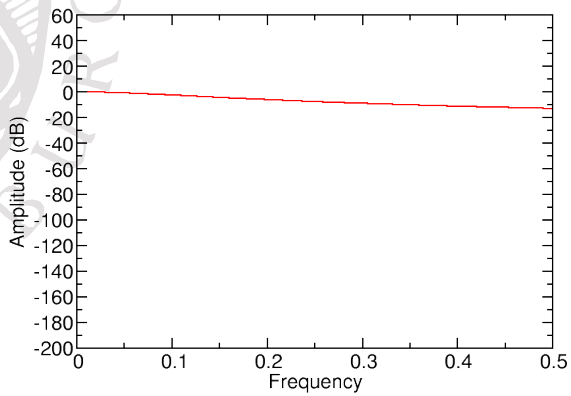
Further Reading

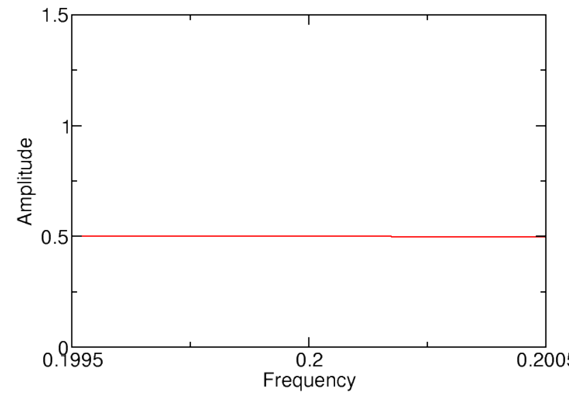
Digital Signal Processing:
A Practical Guide for Engineers and Scientists,
Steven W.Smith, Newnes, 2003
http://www.dspguide.com



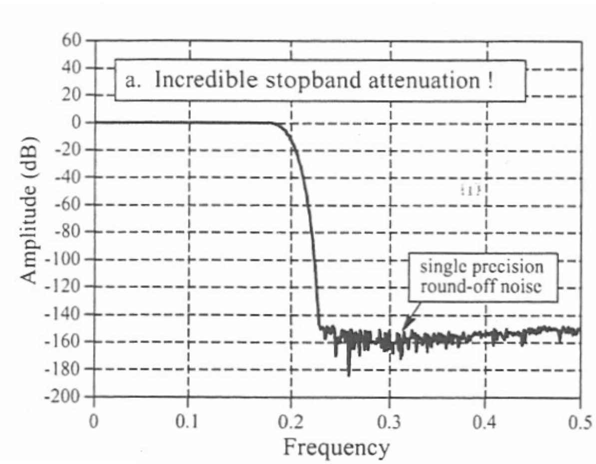
Digital Signal Processing: the perfect filter?

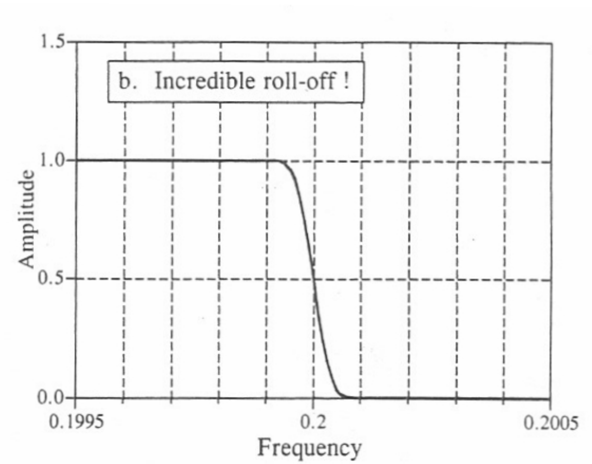
Example





low pass
RC Filter





from Smith, *Digital Signal Processing*

- 32,001 point *windowed-sinc* filter configured as a low pass filter
- Stopband attenuation >120dB (1 part in a million)
- Roll off 0.0125% of sample rate
- 1kHz low pass filter gain 1+/-0.002 d.c. to 1000Hz
  gain <0.0002 above 1001Hz

Moving Average Filter

- Commonly used digital filter
  easy to use and understand
- *Optimum* filter for reducing random noise *and* minimising degradation of step response
- Good signal smoother (time domain)
- Poor filter (frequency domain)
- Noise reduction $\propto \sqrt{m}$

$$y(i) = \frac{1}{m}\sum_{j=0}^{m-1} x(i+j)$$

where *x* is the input signal, *y* the output signal and *m* is the number of points (samples) in the average.

Alternatively, symmetrically about the output point

$$y(i) = \frac{1}{m}\sum_{j=-(m-1)/2}^{(m-1)/2} x(i+j) \qquad m \text{ odd}$$

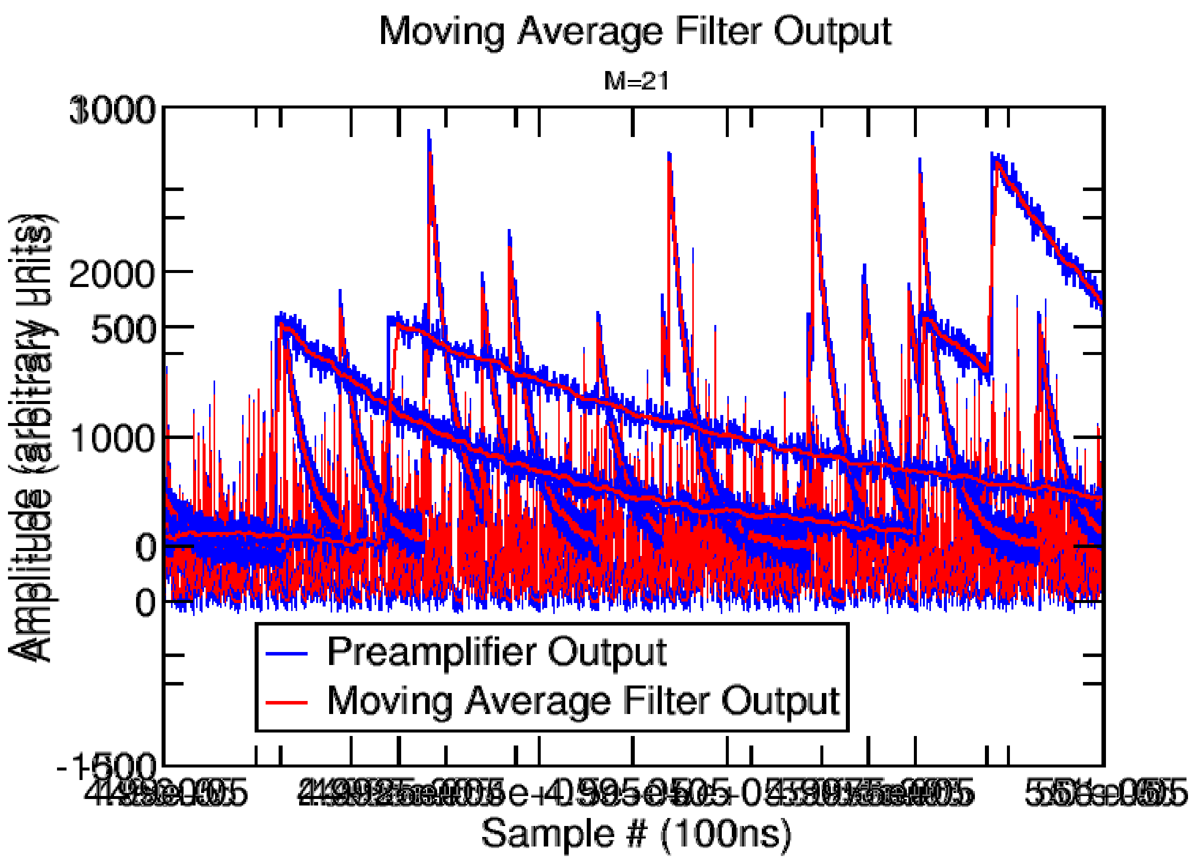
- No relative shift between input and output

DSP Program: Moving Average Filter

```
      PROGRAM ma

C     Number of samples = n
      INTEGER n
      PARAMETER (n = 1000000)

C     Moving average sample length
C     = m samples ( m is an odd number )
      INTEGER m
      PARAMETER (m=21)

      INTEGER i, j
      REAL t(0:n-1), x(0:n-1), y(0:n-1)

C     Read input data
      DO i = 0, n - 1
       READ( 5, * ) t( i ), x( i )
      ENDDO

C     Calculate moving average
C
C     Loop for each data point
C     Zero output data point
C     Loop m times for sum
C     Calculate m-point average

      DO i = ( m - 1 ) / 2, n - ( m - 1 ) / 2
       y( i ) = 0.0
       DO j = - ( m - 1 ) / 2, ( m - 1 ) / 2
        y( i ) = y( i ) + x( i + j )
       ENDDO
       y( i ) = y( i ) / m
      ENDDO

C     Write moving average filter output
      DO i = ( m - 1 ) / 2, n - ( m - 1 ) / 2
       WRITE( 6, * ) t( i ), y( i )
      ENDDO

      STOP
      END
```

FORTRAN77 source code

Moving Average Filter Output



Moving Average Filter

Implementation by Recursion

Consider two adjacent output points produced by a 5-point moving average filter, for example,

$$y(50) = x(48) + x(49) + x(50) + x(51) + x(52)$$
$$y(51) = x(49) + x(50) + x(51) + x(52) + x(53)$$

or,

$$y(51) = y(50) + x(53) - x(48)$$

More generally,

$$y(i) = y(i-1) + x(i+p) - x(i-q) \qquad p = \frac{m-1}{2}, q = p+1$$

Note that only two calculations per output data point are required independent of the number of points m in the moving average.