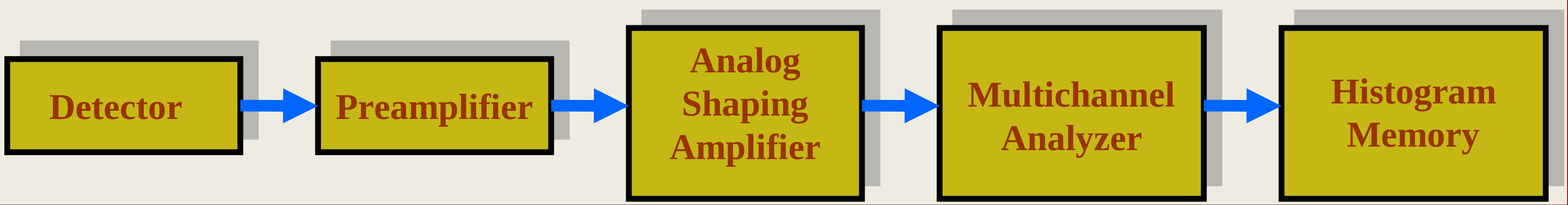
Digital Filters in Radiation Detection and Spectroscopy

Digital Radiation Measurement and Spectroscopy

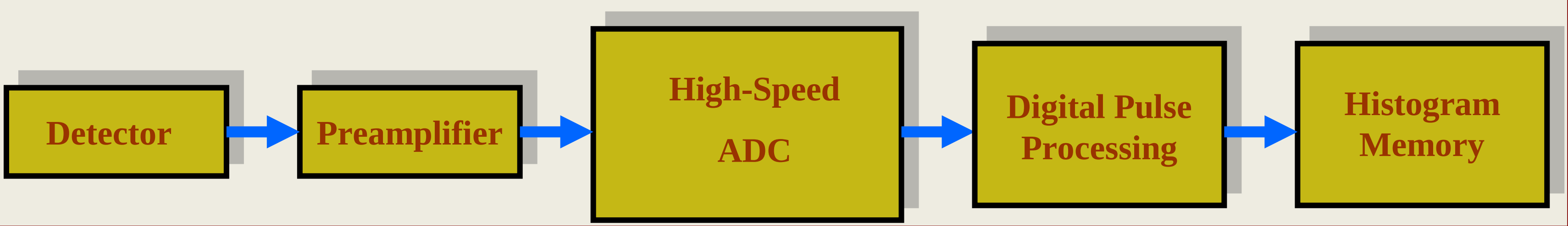
NE/RHP 537

Classical and Digital Spectrometers

- Classical Spectrometer

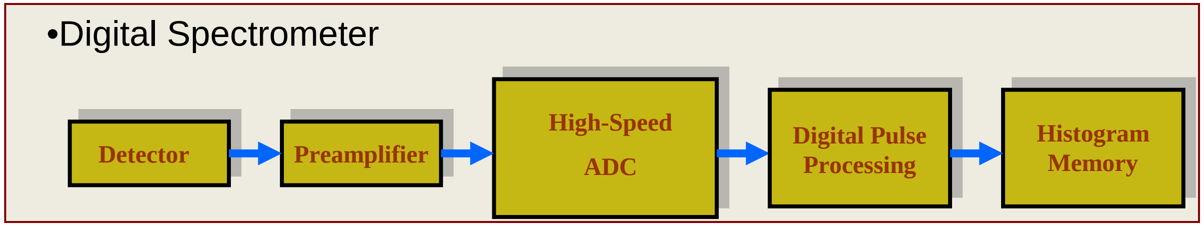


- Digital Spectrometer

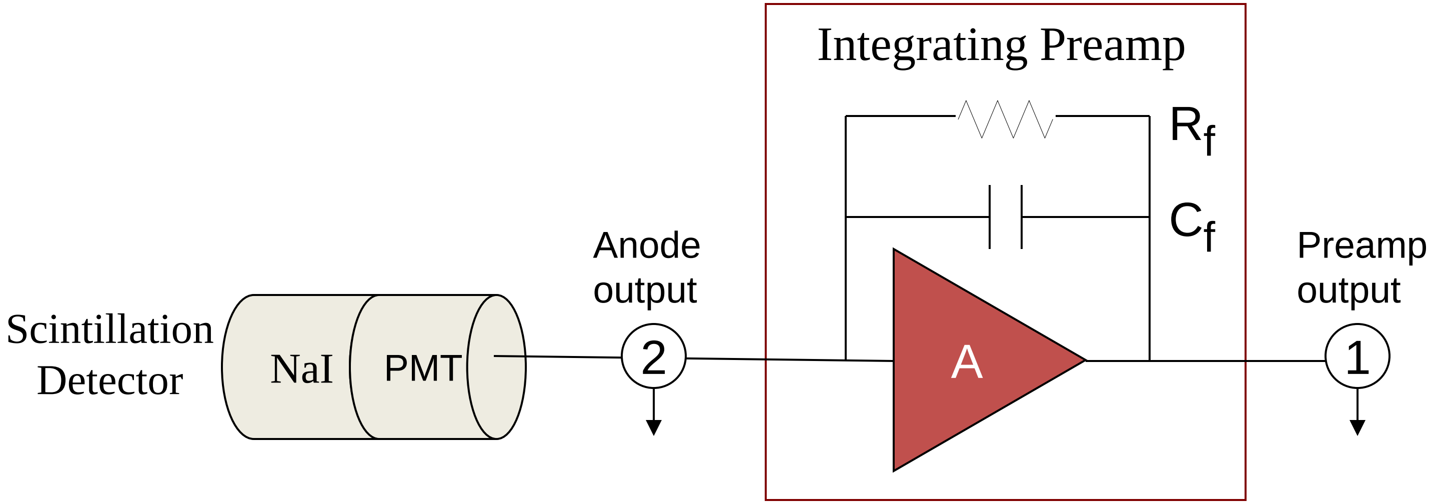


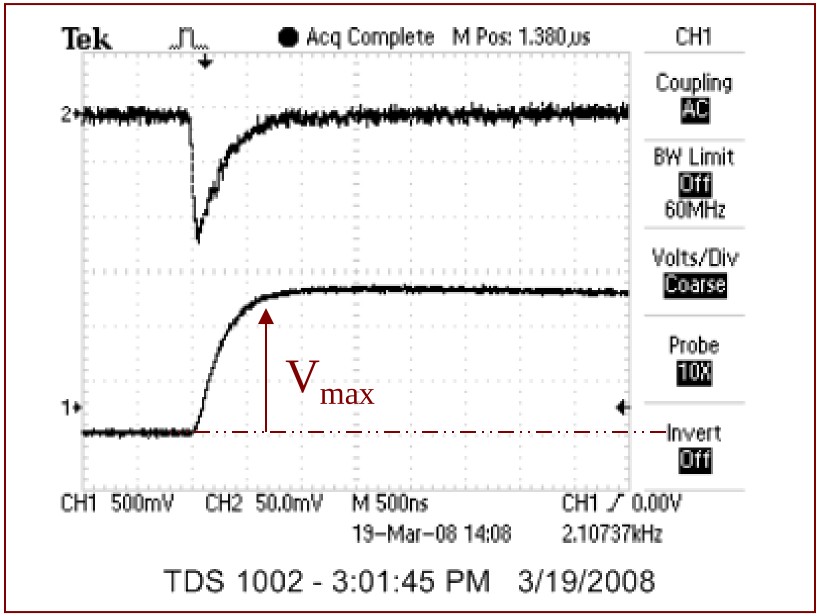
Digital Spectrometers: Advantages

- Pulse processing algorithm is easy to edit
- No bulky analog electronics
- Post-processing
- Digital pulse shaping
- More cost-effective
- The algorithm is stable and reliable, no thermal noise or other fluctuations
- Effects, such as pile-up can be corrected or eliminated at the processing level
- Signal capture and processing can be based more easily on coincidence criteria between different detectors or different parts of the same detector
- **Disadvantage**: Lack of expert in (Digital Systems + Nuclear Spectroscopy)



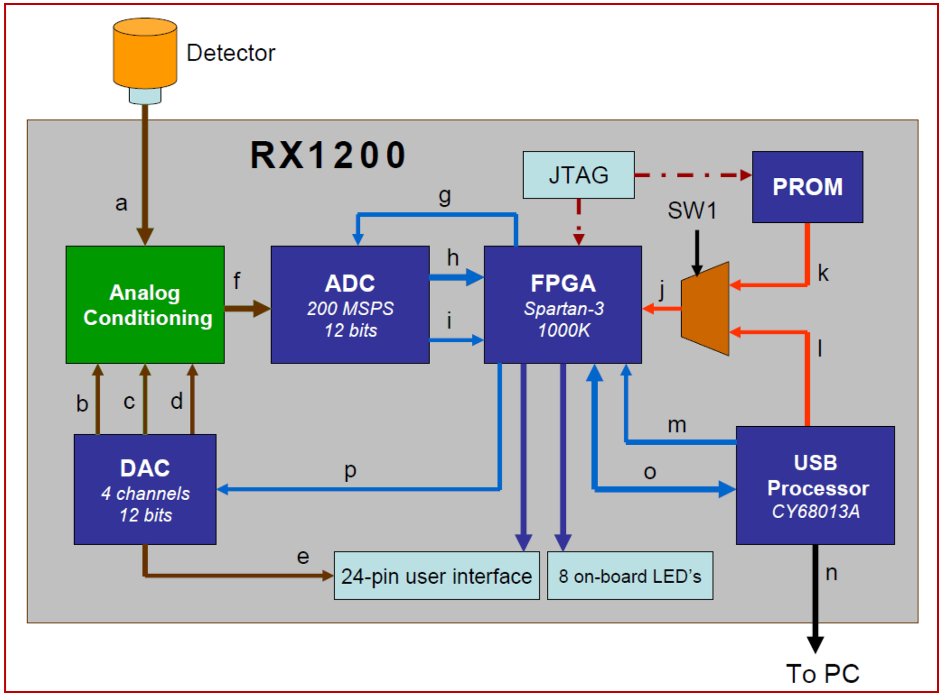
Signal Pulses From Scintillation Detectors





Digital Spectrometers: Digital Pulse Processor (Hardware)

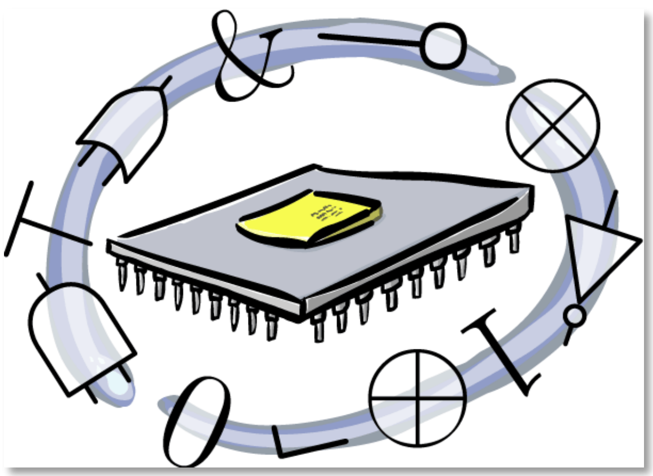
- Analog Conditioning:
  - Nyquist filter ($F_{max} < \frac{F_{adc}}{2}$)
  - Offset adjustment
  - Amplification Gain
- High resolution, fast Analog-to-Digital Convertor (ADC)
- Processing Device:
  - FPGA (Field-Programmable Gate Array)
  - DSP (Digital Signal Processor)
- Interface:
  - USB, PCI, Ethernet

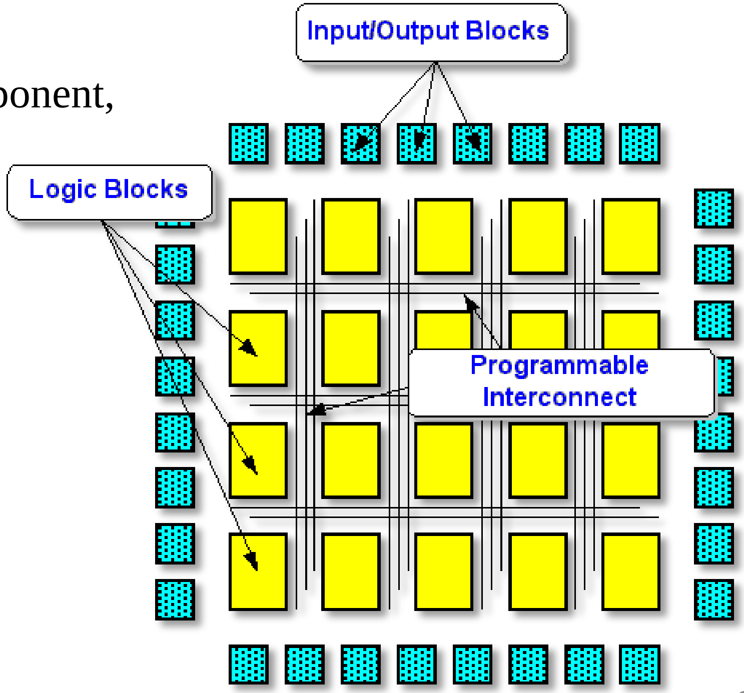


A typical FPGA-based digital pulse processor for scintillation detector

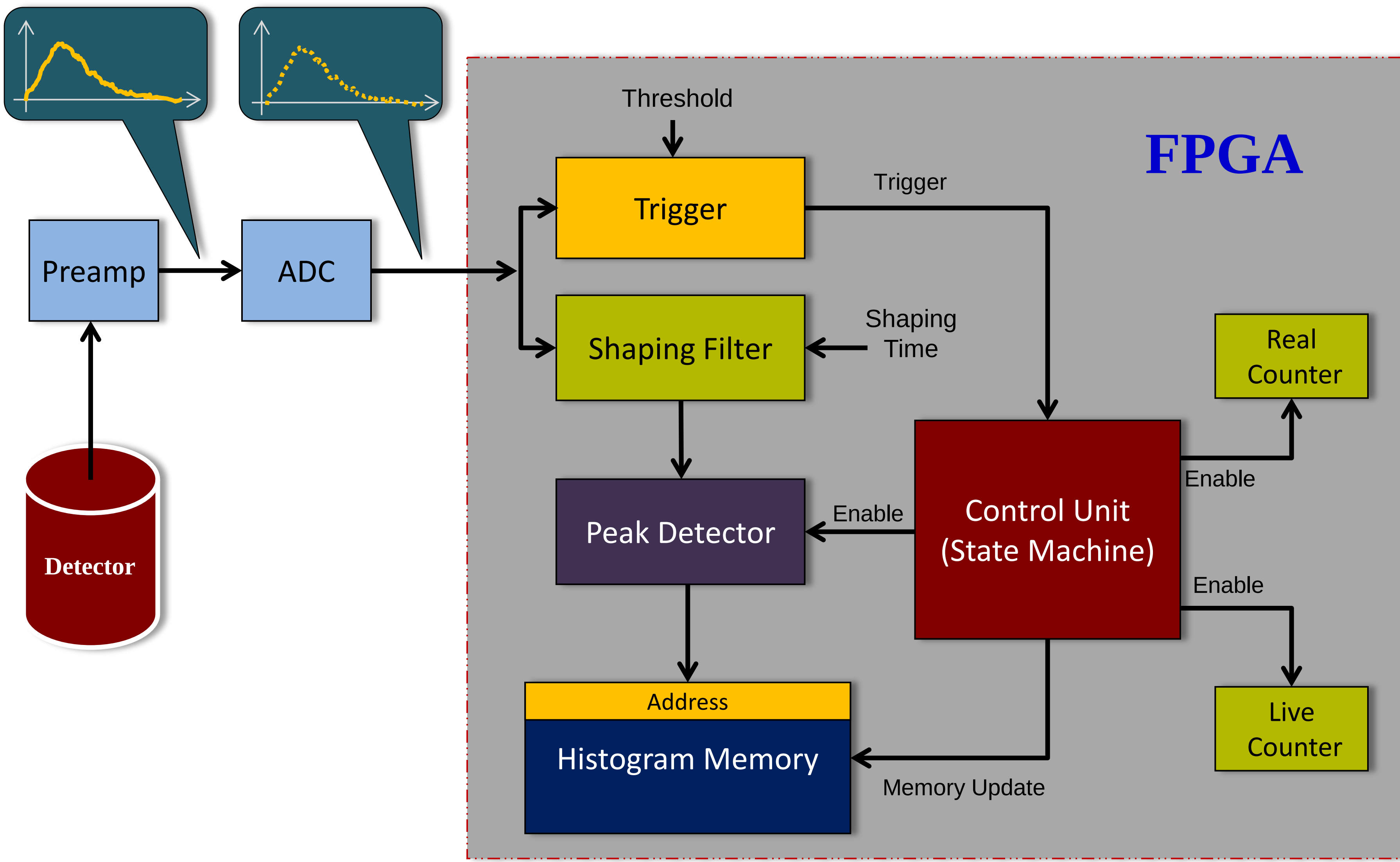
Digital Spectrometers: Field-Programmable Gate Array (FPGA)

- FPGAs are an array of programmable logic cells interconnected by a network of wires and configurable switches.
- A FPGA has a large number of these cells available to form multipliers, adders, accumulators and so forth in complex digital circuits.
- FPGAs can be infinitely reprogrammed in-circuit in only a small fraction of a second.
- New FPGA devices provide integrated memory component, to be used as histogram memory.

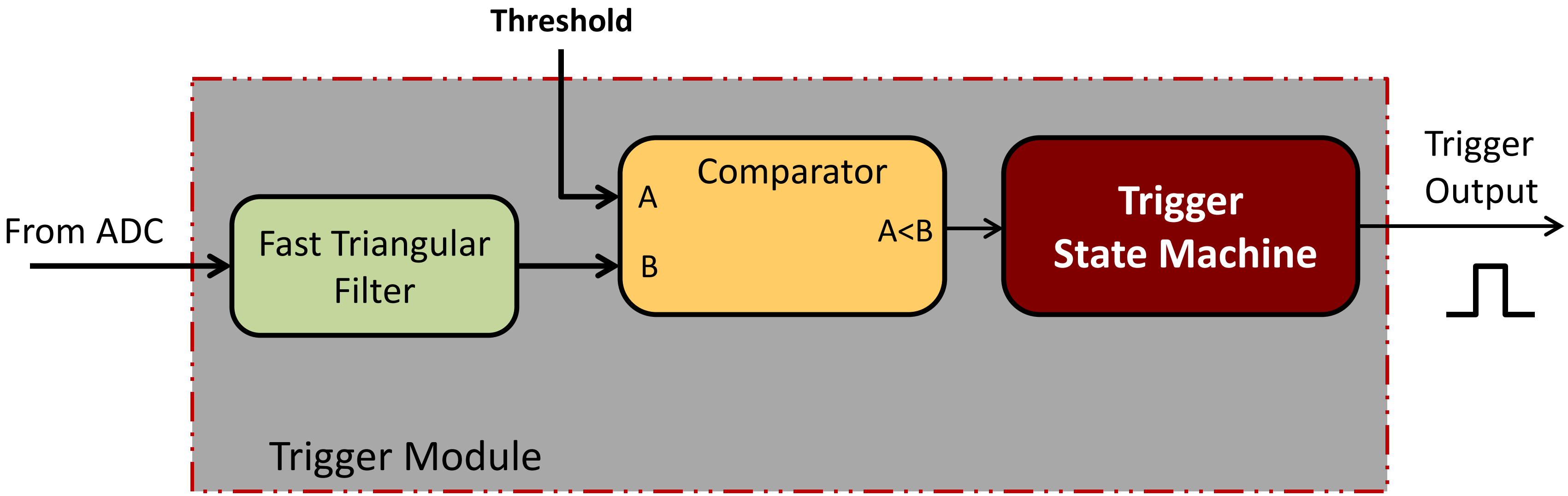




Digital Spectrometers: Inside the FPGA

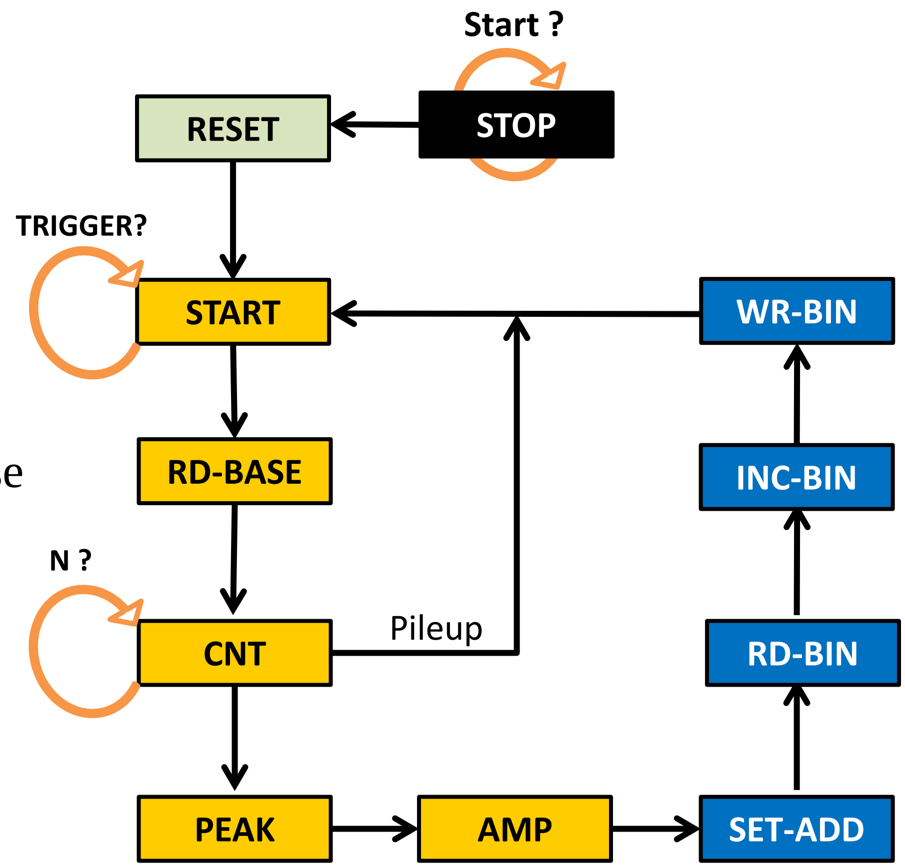


Digital Spectrometers: Trigger Module



Digital Spectrometers: MCA State Machine

1. **STOP:** wait until receiving start pulse
2. **RESET:** start MCA by resetting counters, buffers,…
3. **START:** wait for a trigger pulse
4. **RD-BASE:** read baseline
5. **CNT:** wait for N clock cycles
6. **PEAK:** take a sample from the shaped pulse (peak)
7. **AMP:** calculate $\boldsymbol{amp} = \boldsymbol{peak} - \boldsymbol{baseline}$
8. **SET-ADD:** locate and set memory address which corresponds to the pulse amplitude
9. **RD-BIN:** read current counts from the located address
10. **INC-BIN:** increment the count by one
11. **WR-BIN:** write the incremented count to the same address, go to START state
12. **START:** wait for another trigger



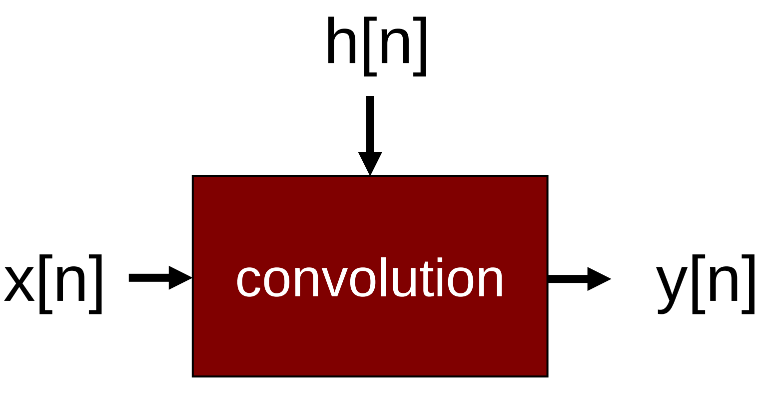
MCA State Machine with 11 states

Digital Filters: Finite Impulse Response (FIR)

Convolution Operation

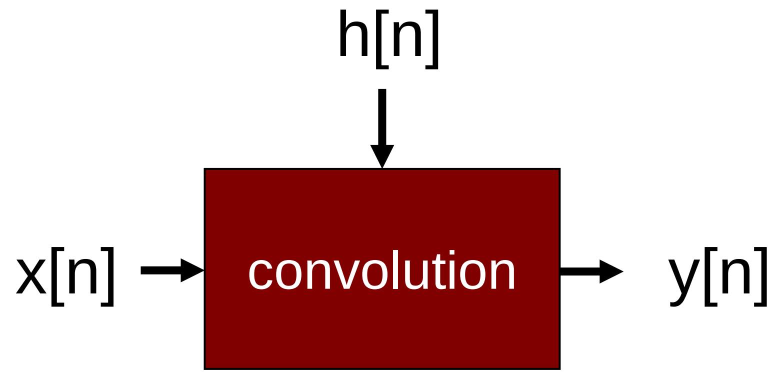
- A finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely.

- For FIR digital filters, the input-output relation involves a finite sum of products
- Convolution operation defines how the input signal is related to the output signal



Digital Filters: Finite Impulse Response (FIR)

Convolution Operation (Sum of Products)



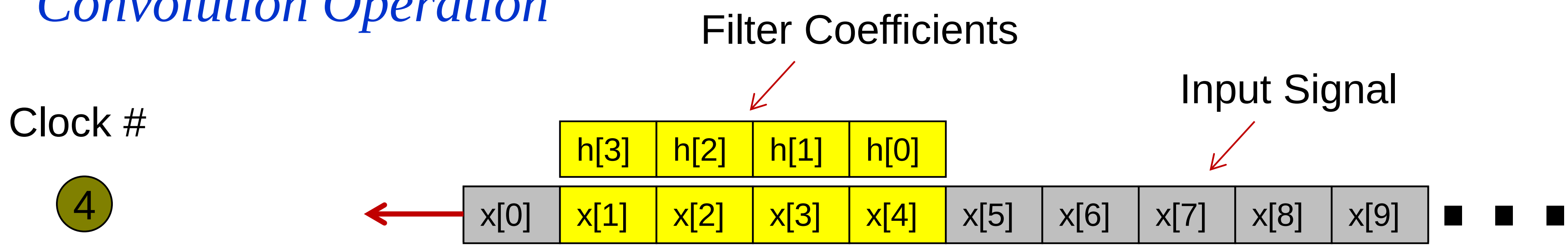
- x[n] is the input signal
- y[n] is the output signal
- h[n] is impulse response (filter coefficients)
- n is the sample number
- N+1 is the filter size

$$y[n] = \sum_{k=0}^{N} h[k] \,.\, x[n-k]$$

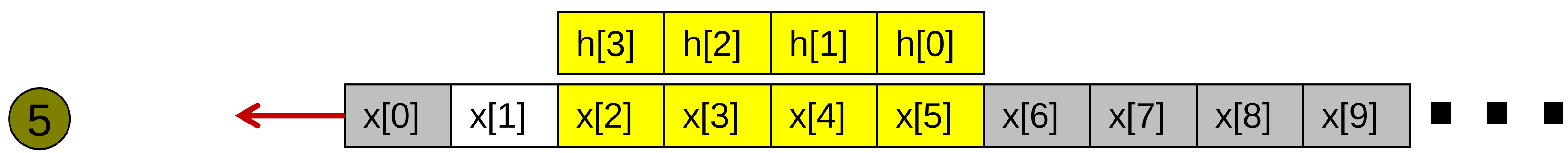
$$y[n] = h[0].x[n] + h[1].x[n-1] + \ldots + h[N].x[n-N]$$

Digital Filters: Finite Impulse Response (FIR)

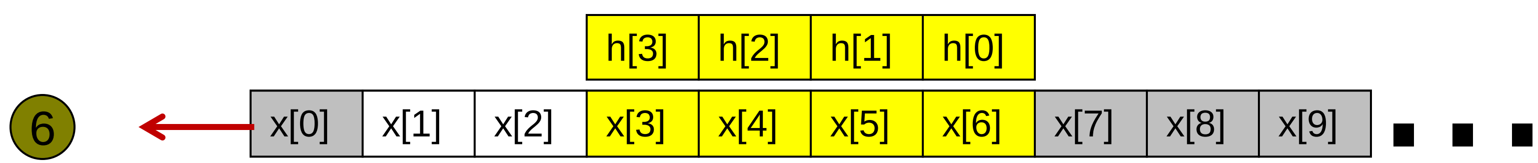
Convolution Operation



Output Signal: $y[4] = h[0].x[4] + h[1].x[3] + h[2].x[2] + h[3].x[1]$



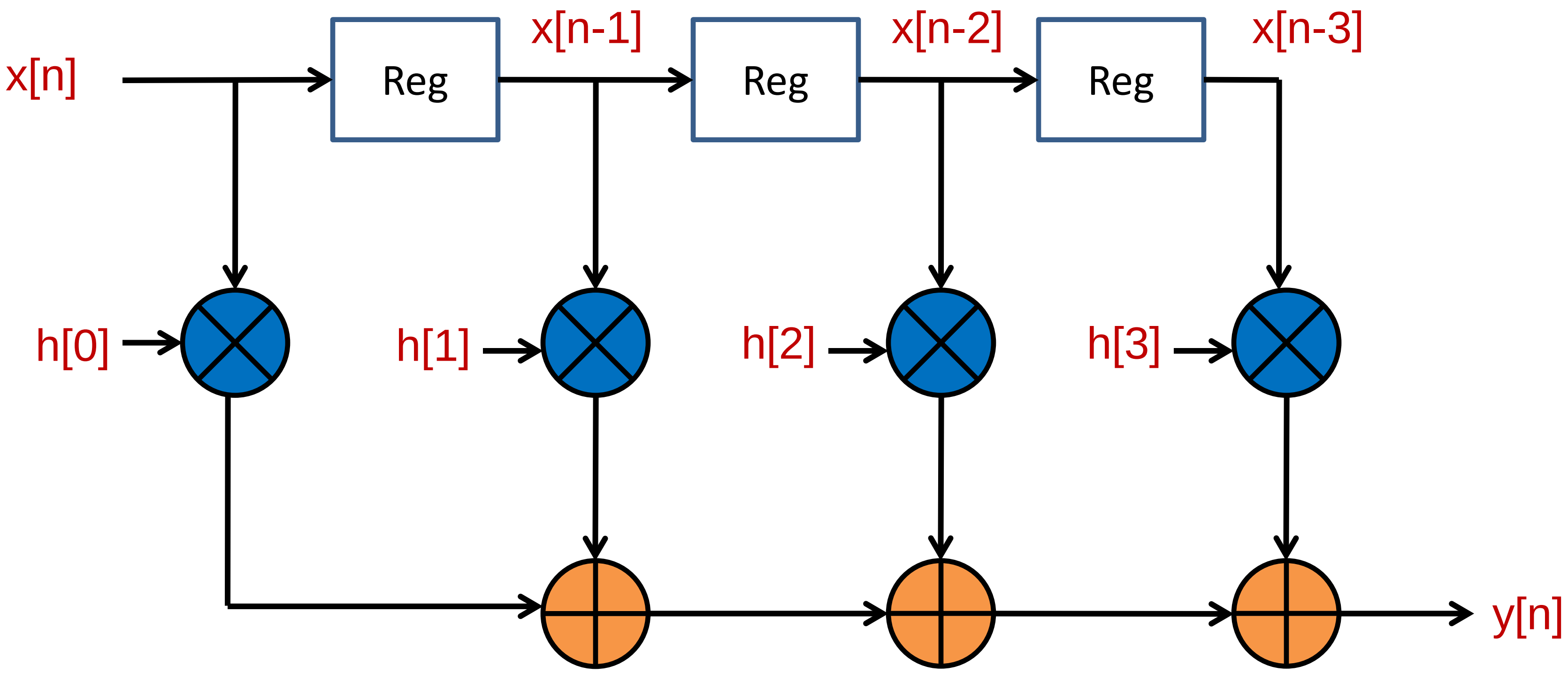
Output Signal: $y[5] = h[0].x[5] + h[1].x[4] + h[2].x[3] + h[3].x[2]$



Output Signal: $y[6] = h[0].x[6] + h[1].x[5] + h[2].x[4] + h[3].x[3]$

Digital Filters: Finite Impulse Response (FIR)

Convolution in Hardware (Realization)



$$y[n] = h[0].x[n] + h[1].x[n-1] + h[2].x[n-2] + h[3].x[n-3]$$

Digital Filters: Finite Impulse Response (FIR)

Moving Average Filter: Noise Reduction

- Consider a digital filter whose output signal y[n] is the average of the four most recent values of the input signal x[n]:

$$y[n] = \frac{1}{4} \left(x[n] + x[n-1] + x[n-2] + x[n-3] \right)$$

- Such a filter is referred to as a Moving Average Filter and is commonly used for noise reduction.
- The amount of noise reduction is equal to the square-root of the number of points in the average. For example, a 100 point moving average filter reduces the noise by a factor of 10.

Digital Filters: Finite Impulse Response (FIR)

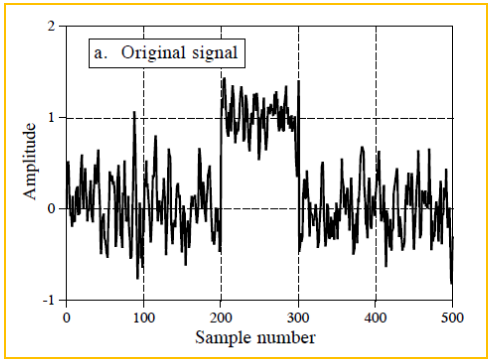
Moving Average Filter: Noise Reduction

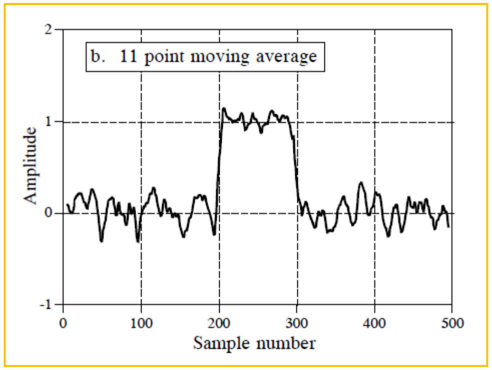
- Example of a moving average filter.

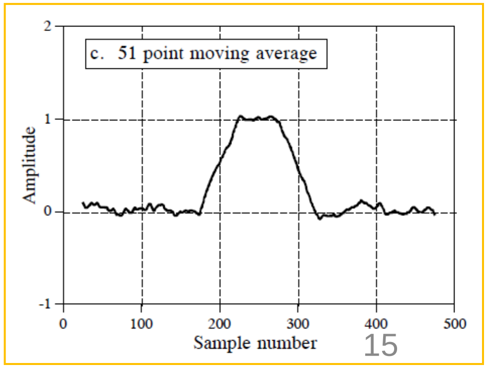
- In (a), a rectangular pulse is buried in random noise.

- In (b) and (c), this signal is filtered with 11 and 51 point moving average filters, respectively.

- As the number of points in the filter increases, the noise becomes lower; however, the edges becoming less sharp.







Digital Filters: Finite Impulse Response (FIR)

Moving Average Filter: Noise Reduction

But how to implement a Moving Average Filter?

$$y[n] = \tfrac{1}{4} \left(x[n] + x[n-1] + x[n-2] + x[n-3] \right)$$

Recall the convolution operation (sum of products)

$$\boxed{y[n] = h[0].x[n] + h[1].x[n-1] + h[2].x[n-2] + h[3].x[n-3]}$$

In this example, h[n] should be:

$$h = [\ \tfrac{1}{4}\ ,\ \tfrac{1}{4}\ ,\ \tfrac{1}{4}\ ,\ \tfrac{1}{4}\ ]$$

OR

h = [1/N , 1/N , 1/N , 1/N] , where N is the number of coefficients in the filter

Digital Filters: Finite Impulse Response (FIR)

Trapezoidal Filter: Pulse Shaping (for Pre-Amp pulses)

Positive Averaging over L samples

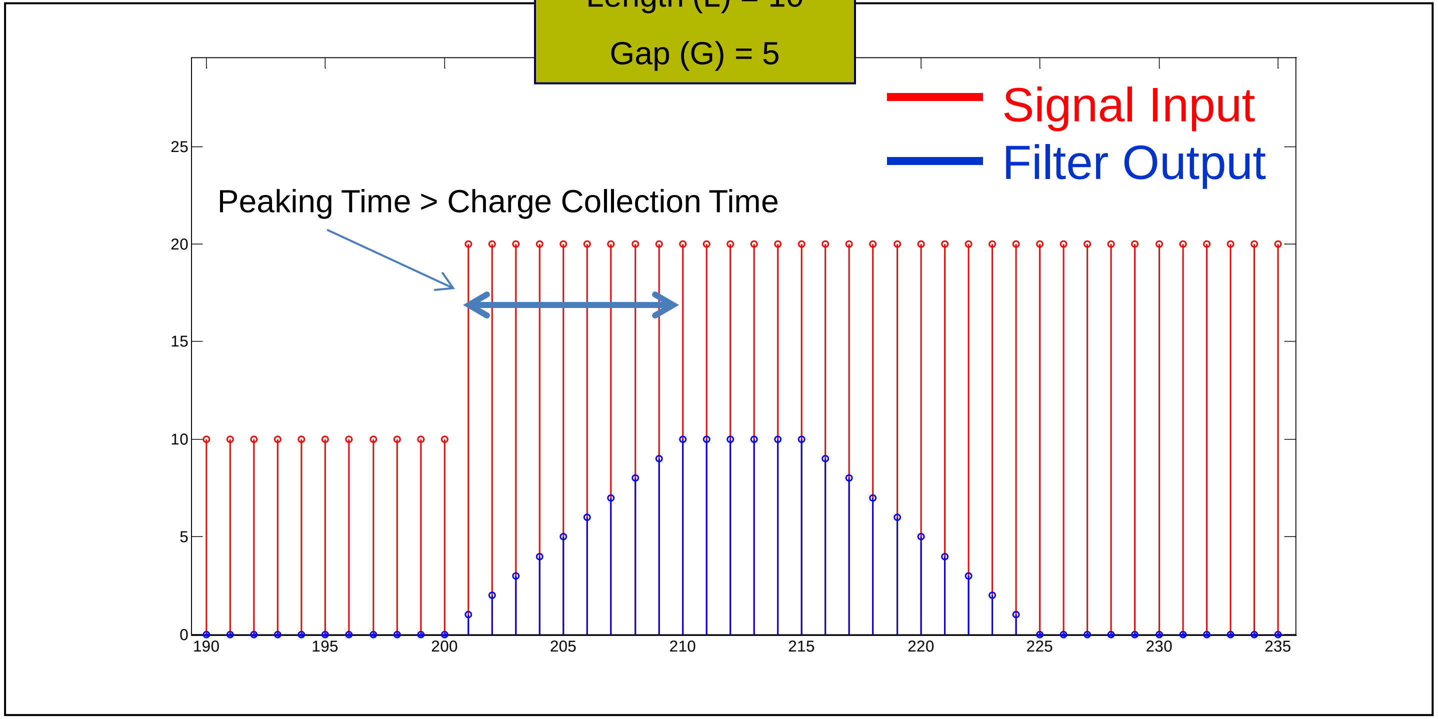
Gap

Negative Averaging over L samples

h = [.1, .1, .1, .1, .1, .1, .1, .1, .1, .1, 0, 0, 0, 0, 0, -.1, -.1, -.1, -.1, -.1, -.1, -.1, -.1, -.1, -.1]

Length (L) = 10

Gap (G) = 5

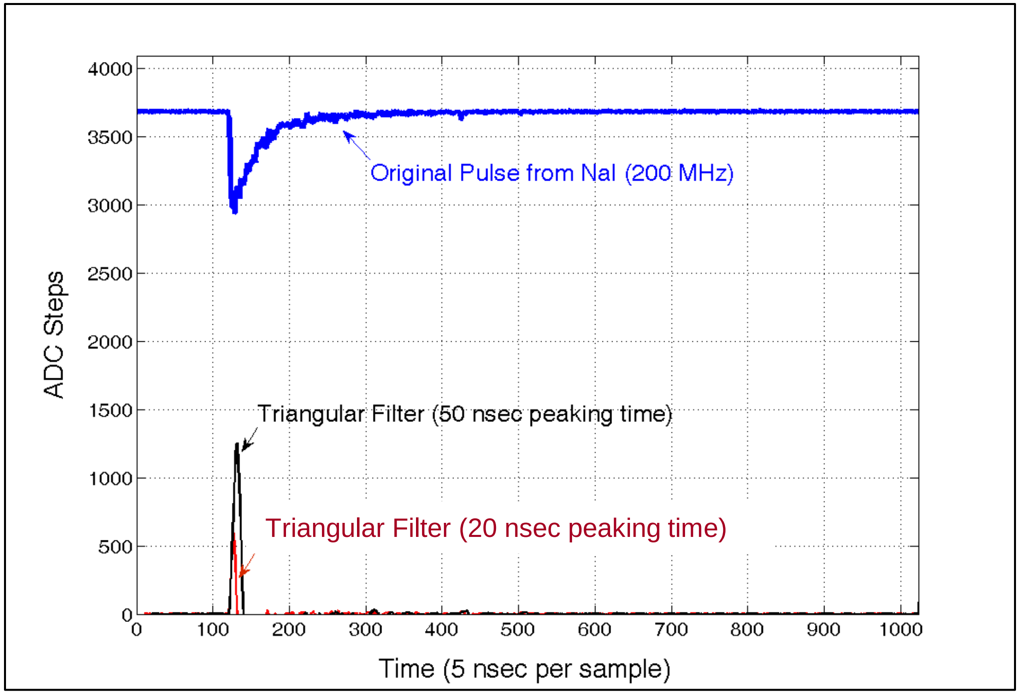


Digital Filters: Finite Impulse Response (FIR)

Triangular Filter: Trigger

- A triangular filter is a special trapezoidal filter with G=0.

- Triangular filters with peaking time of 25 nsec and 50 nsec (with T=5 nsec) are:
  - h1 = [-1, -1, -1, -1, -1, 1, 1, 1, 1, 1]
  - h2 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1 , 1, 1, 1, 1, 1]



Digital Filters: Finite Impulse Response (FIR)

Triangular Filter: Pulse Integration (for Anode pulses)

Integration Time = L . Sampling Period ;

(The time over which scintillator emits most of its light ~ 99% ~ 4.6 decay time)

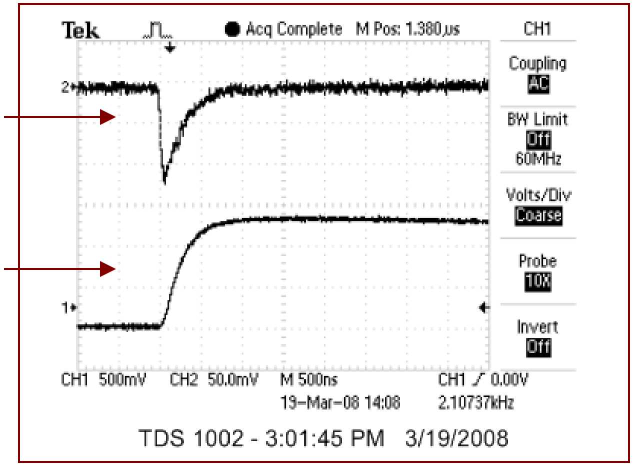
NaI: 230 * 4.6 = 1058 nsec

•G=0 (Triangle Filter)

•Unity Coefficients (no normalization)

Anode output

Preamp output



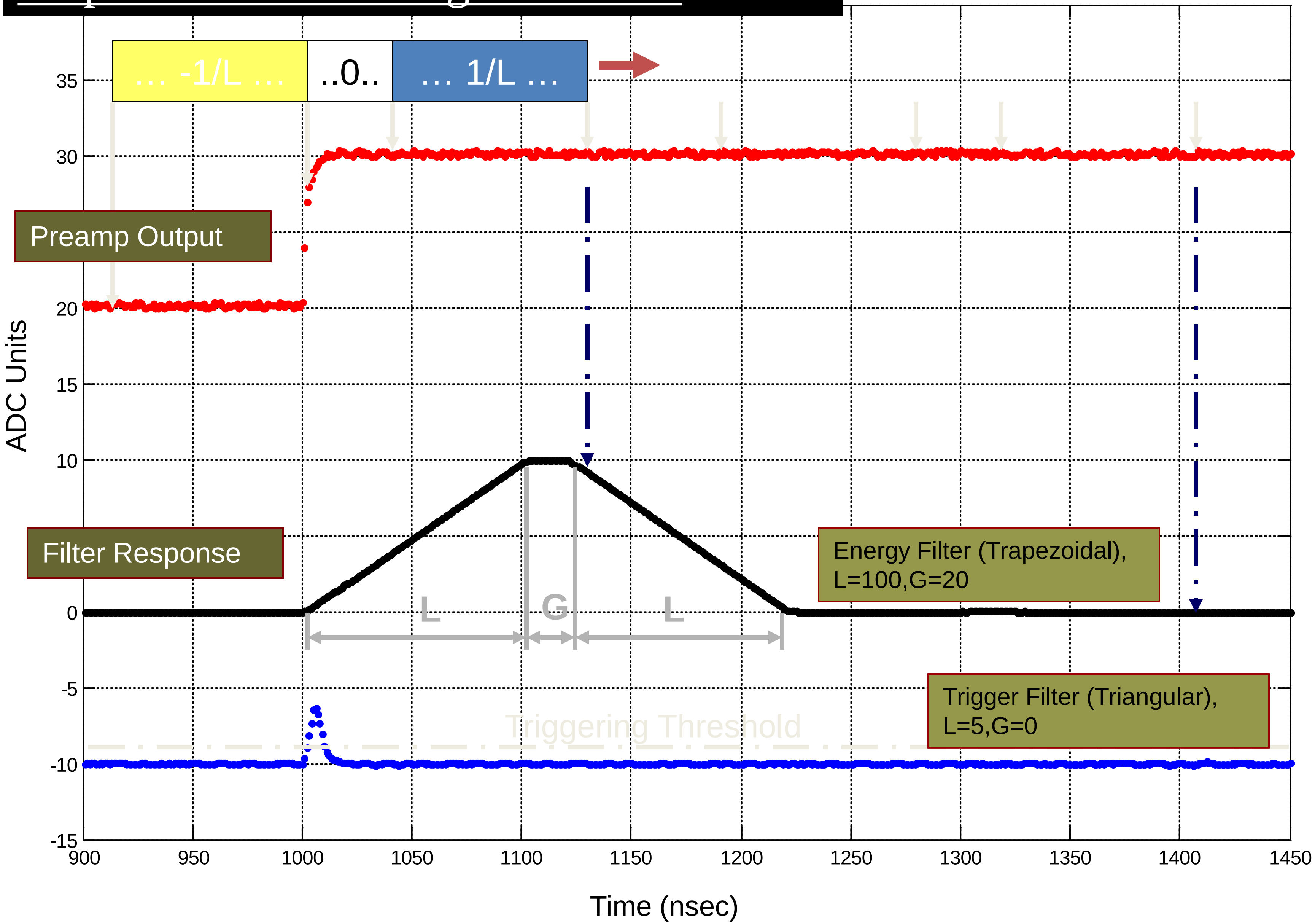
L L

For pulses with negative polarity: … -1 … | … 1 …

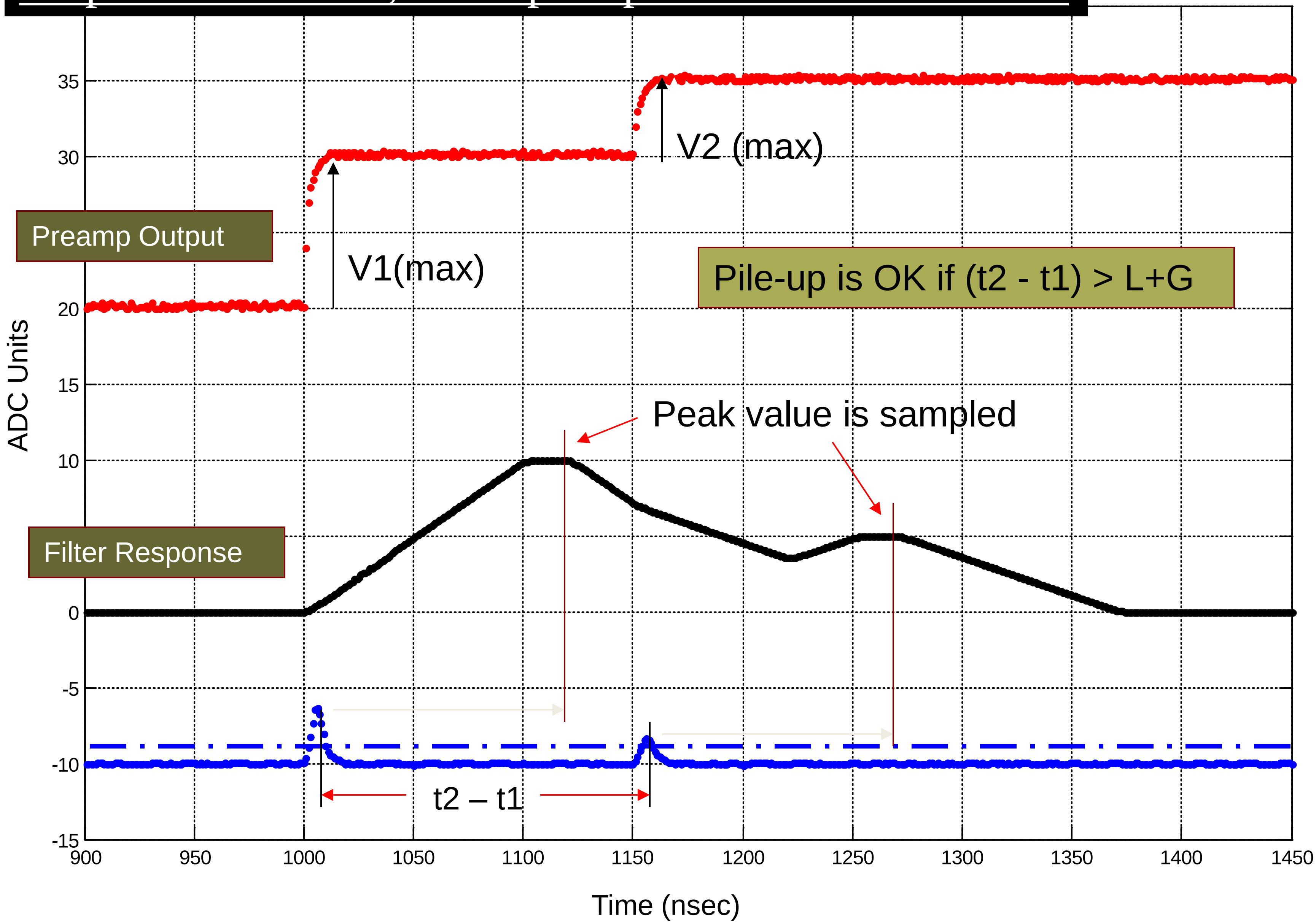
Pulse Integration | Baseline Correction

For pulses with positive polarity: … 1 … | … -1 …

Trapezoidal and Triangular Filters



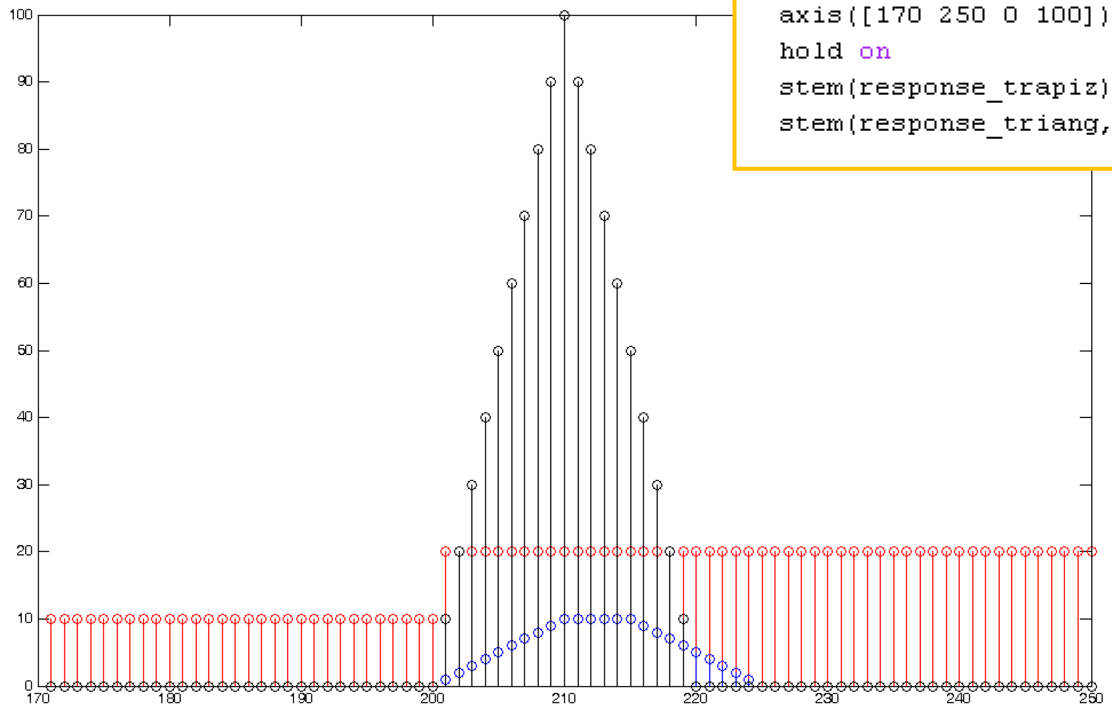
Trapezoidal Filters, Pile-up Inspection and Correction



Digital Filters: Finite Impulse Response (FIR)

Trapezoidal and Triangular Filter: Response to Step Function

```
% Trapezoidal Filters
step = [ones(1,200)*10 ones(1,200)*20];
filter_trapiz = [ones(1,10)/10 zeros(1,5) ones(1,10)/-10];
filter_triang = [ones(1,10) -ones(1,10)];
response_trapiz = conv(step, filter_trapiz);
response_triang = conv(step, filter_triang);
stem(step, 'r');
axis([170 250 0 100]);
hold on
stem(response_trapiz);
stem(response_triang,'k');
```



Digital Filters: Finite Impulse Response (FIR)

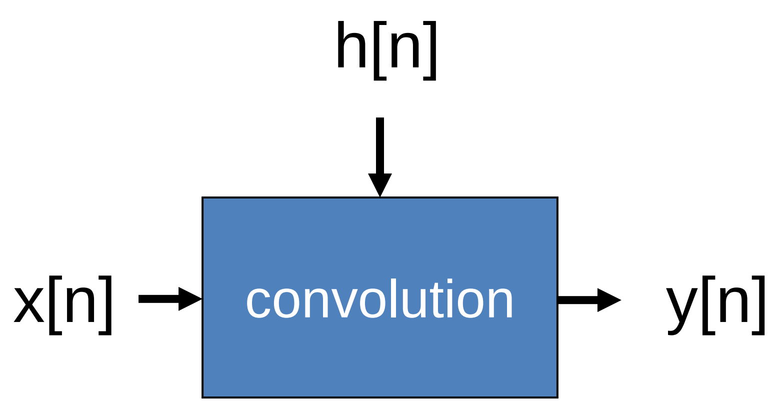
Some Convolution Practices (with pen and paper!)

Pre-Amp Pulse: x = [1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3]

Digital Filter: h = [½, ½, 0, 0, -½, -½]

Using the Convolution equation, find y[5-11].

$$y[n] = \sum_{k=0}^{N} h[k] \,.\, x[n-k]$$



Digital Filters: Finite Impulse Response (FIR)

Some Convolution Practices (with pen and paper!)

Y[5] = (h[0].x[5-0]) + (h[1].x[5-1]) + (h[2].x[5-2]) + (h[3].x[5-3]) + (h[4].x[5-4]) + (h[5].x[5-5])

Y[5] = (h[0].x[5]) + (h[1].x[4]) + (h[2].x[3]) + (h[3].x[2]) + (h[4].x[1]) + (h[5].x[0])

Y[5] = (0.5 * 1) + (0.5 * 1) + (0 * 1) + (0 * 1) + (-0.5 * 1) + (-0.5 * 1) = 0

Y[6] = (h[0].x[6-0]) + (h[1].x[6-1]) + (h[2].x[6-2]) + (h[3].x[6-3]) + (h[4].x[6-4]) + (h[5].x[6-5])

Y[6] = (h[0].x[6]) + (h[1].x[5]) + (h[2].x[4]) + (h[3].x[3]) + (h[4].x[2]) + (h[5].x[1])

Y[6] = (0.5 * 3) + (0.5 * 1) + (0 * 1) + (0 * 1) + (-0.5 * 1) + (-0.5 * 1) = 1

Y[7] = (h[0].x[7-0]) + (h[1].x[7-1]) + (h[2].x[7-2]) + (h[3].x[7-3]) + (h[4].x[7-4]) + (h[5].x[7-5])

Y[7] = (h[0].x[7]) + (h[1].x[6]) + (h[2].x[5]) + (h[3].x[4]) + (h[4].x[3]) + (h[5].x[2])

Y[7] = (0.5 * 3) + (0.5 * 3) + (0 * 1) + (0 * 1) + (-0.5 * 1) + (-0.5 * 1) = 2

Digital Filters: Finite Impulse Response (FIR)

Some Convolution Practices (with pen and paper!)

Y[8] = (h[0].x[8-0]) + (h[1].x[8-1]) + (h[2].x[8-2]) + (h[3].x[8-3]) + (h[4].x[8-4]) + (h[5].x[8-5])

Y[8] = (h[0].x[8]) + (h[1].x[7]) + (h[2].x[6]) + (h[3].x[5]) + (h[4].x[4]) + (h[5].x[3])

Y[8] = (0.5 * 3) + (0.5 * 3) + (0 * 3) + (0 * 1) + (-0.5 * 1) + (-0.5 * 1) = 2

Y[9] = (h[0].x[9-0]) + (h[1].x[9-1]) + (h[2].x[9-2]) + (h[3].x[9-3]) + (h[4].x[9-4]) + (h[5].x[9-5])

Y[9] = (h[0].x[9]) + (h[1].x[8]) + (h[2].x[7]) + (h[3].x[6]) + (h[4].x[5]) + (h[5].x[4])

Y[9] = (0.5 * 3) + (0.5 * 3) + (0 * 3) + (0 * 3) + (-0.5 * 1) + (-0.5 * 1) = 2

Y[10] =(h[0].x[10-0]) + (h[1].x[10-1]) + (h[2].x[10-2]) + (h[3].x[10-3]) + (h[4].x[10-4]) + (h[5].x[10-5])

Y[10] = (h[0].x[10]) + (h[1].x[9]) + (h[2].x[8]) + (h[3].x[7]) + (h[4].x[6]) + (h[5].x[5])

Y[10] = (0.5 * 3) + (0.5 * 3) + (0 * 3) + (0 * 3) + (-0.5 * 3) + (-0.5 * 1) = 1

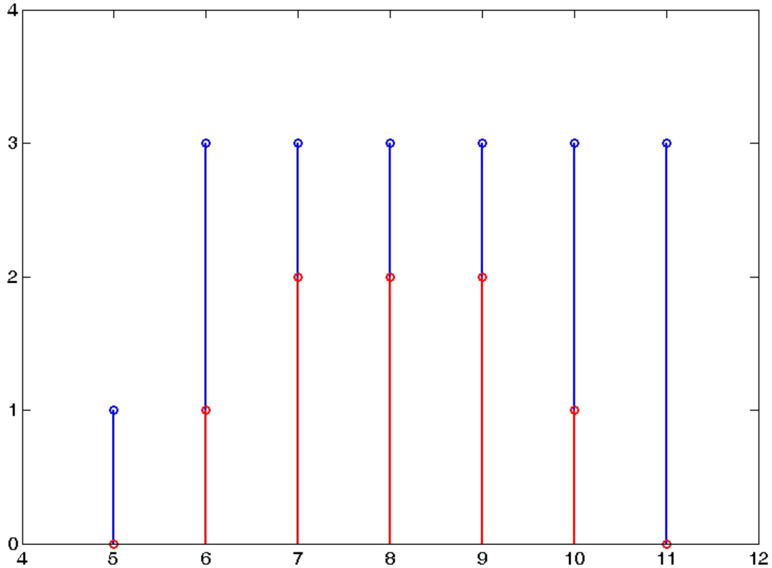
Digital Filters: Finite Impulse Response (FIR)

Some Convolution Practices (with pen and paper!)

Y[11] =(h[0].x[11-0]) + (h[1].x[11-1]) + (h[2].x[11-2]) + (h[3].x[11-3]) + (h[4].x[11-4]) + (h[5].x[11-5])

Y[11] = (h[0].x[11]) + (h[1].x[10]) + (h[2].x[9]) + (h[3].x[8]) + (h[4].x[7]) + (h[5].x[6])

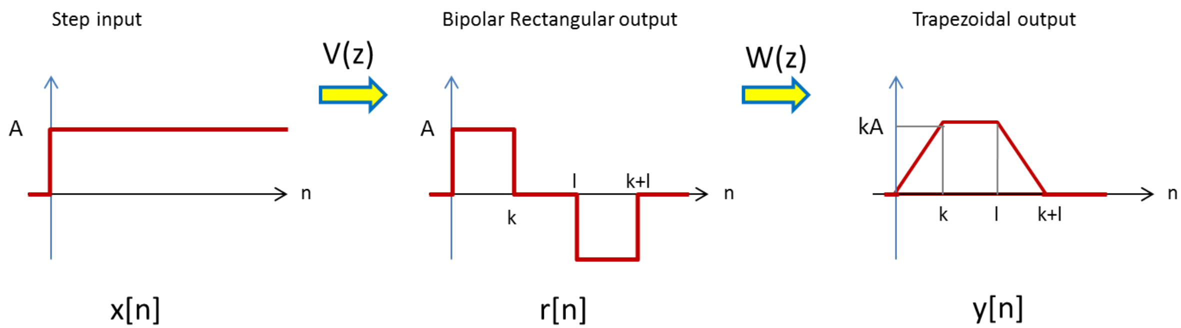
Y[11] = (0.5 * 3) + (0.5 * 3) + (0 * 3) + (0 * 3) + (-0.5 * 3) + (-0.5 * 3) = 0

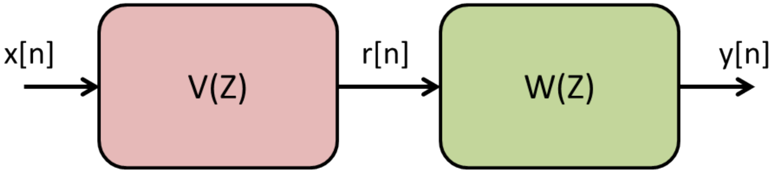


Digital Filters: Infinite Impulse Response (IIR)

Trapezoidal shaping using recursive algorithm

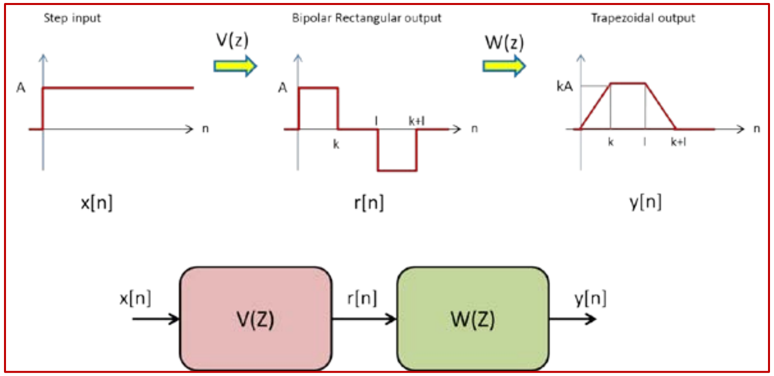
- Digital filters can be realized in FPGA with minimum resources using recursive algorithm.
- In a recursive algorithm, to synthesis a trapezoidal output y[n] from a step input x[n], we can process the input in two steps.
- In the first step, the step input x[n] is first converted to a bipolar rectangular pulse r[n]. In the second step, r[n] is converted to a trapezoidal output using an accumulator.





Digital Filters: Infinite Impulse Response (IIR)

Trapezoidal shaping using recursive algorithm



$$Z\ transform:\ H(z) = \sum_{k=-\infty}^{\infty} h[k] z^{-k}$$

$$x[n] = A\,u[n]$$

$$X(z) = \frac{A}{1 - z^{-1}}$$

$$r[n] = A\,\{u[n] - u[n-k] - u[n-l] + u[n-l-k]\}$$

$$R(z) = A\left\{\frac{1}{1 - z^{-1}} - \frac{z^{-k}}{1 - z^{-1}} - \frac{z^{-l}}{1 - z^{-1}} + \frac{z^{-l-k}}{1 - z^{-1}}\right\}$$

$$R(z) = \frac{A}{1 - z^{-1}}\{1 - z^{-k} - z^{-l} + z^{-l-k}\}$$

$$r[n] = x[n] * v[n]$$

$$V(z) = \frac{R(z)}{X(z)} = 1 - z^{-k} - z^{-l} + z^{-l-k}$$

$$Z\ transform\ of\ an\ accumulator\ = W(z) = \frac{Y(z)}{R(z)} = \frac{1}{1 - z^{-1}}$$

Digital Filters: Infinite Impulse Response (IIR)

Trapezoidal shaping using recursive algorithm

- The Z transfer function is: $$H(z) = \frac{Y(z)}{X(z)} = V(z).W(z) = \frac{1 - Z^{-k} - Z^{-l} + Z^{-l-k}}{1 - z^{-1}}$$

- And finally, by a reverse Z transform we can find the recursive algorithm for the trapezoidal shaper:

$$\boldsymbol{y[n] = y[n-1] + (x[n] - x[n-k]) - (x[n-l] - x[n-l-k])}$$