Modelling the Recovery of Pulse Peak Pileup for Implementation in an FPGA for a Nuclear Spectroscopy System

ONYEMAECHI N. OFODILE
Manpower Training and Capacity Development Directorate
Nigeria Atomic Energy Commission
Abuja, NIGERIA
onofodile@nigatom.org.ng, http://www.nigatom.org.ng

MATTHEW N. AGU
Nuclear Power Plant Project
Nigeria Atomic Energy Commission
Abuja, NIGERIA
mnagu@nigatom.org.ng, http://www.nigatom.org.ng

Abstract: A major source of error in radiation measurement is the inaccurate instrument reading such as inaccurate radiation count rate that leads to inaccurate determination of activity of a source and hence, inaccurate values of exposure rate. Inaccurate count rate can be a direct consequence of pulse pileup. The current algorithms for dealing with pulse pileup are to identify pulses that have piled up on top of each other, reject that information, and then analyze only 'clean' pulses. This present work is based on the implementation of a mathematical model of linear equations from a matrix in both hardware and software formats. This implementation is in conjunction with Nelder-Mead direct search algorithm for peak search implementation in an FPGA based signal processing system for pulse pileup recovery. In the design, the FPGA subsystem can be implemented by either a hardware form using Matlab and Xilinx blocks or in a software form using a Mcode block. From the simulations, out of the incident pulses and applying the traditional approach, pileups occurred and only the “clean” pulses are recovered. In our approach, the peaks were detected and recovered up to 100% irrespective of their arrival times even in severe pileup situation as opposed to another work that recovered up to 65% of piled up pulses.

Keywords: Dead Time, Detection, FPGA, Peak, Pileup, Recovery, State Machine, Spectroscopy, Zynq-7000

1 Introduction

Generally, the processing of nuclear radiations to determine the type, energy and intensity of such radiations is adversely affected by the responses of the electronic components which culminate in 3 kinds of errors, namely, noise associated with a detected radiation and its processing, dead time and pile-up losses. Apart from the electronic noise, dead time losses arise due to the time interval required to process a radiation pulse during which another detected radiation pulse cannot be processed. It is assumed that each pulse occurring event is followed by a fixed dead time interval t. Thus, an important source of error comes from this finite time required by the counting electronics to detect and process radiation pulses. During this dead time, the system cannot respond to other photons that hit the detector and these events will not be counted and thus are lost. Pileup losses arise due to the fact that two or more radiation pulses may arrive close to themselves in time with the result that their values overlap and are summed up such that the new summed value does not represent any of the constituent pulses. This constitutes serious distortions to the accuracy of measured pulse values.

The current algorithms for dealing with pulse pileup are to identify pulses that have piled up on top of each other, reject that information, and then analyze only 'clean' pulses. In many applications as much as 80% of information can be lost to the effects of dead time and pulse pileup [1]. A recent approach for the detection of pulse peaks is the Direct Search algorithm in which the interest is in resolving or decomposing a set of overlapping peaks into their separate components. Nelder-Mead modified simplex technique can be used for this purpose [2]. As opposed to more traditional optimization methods that use information about the gradient or higher derivatives to search for an optimal point, a direct search algorithm searches a set of points around the current point, looking for one where the value of the objective function is lower than the value at the current point. The Direct Search algorithm is a preferred option than the other optimization methods because of its simplicity, flexibility, and reliability.

2 Pileup Recovery

A method of recovery of piled up pulses relies on the assumption that the complete signal pulse train can be expressed as a linear combination of single pulses, as shown in Figure. 1.

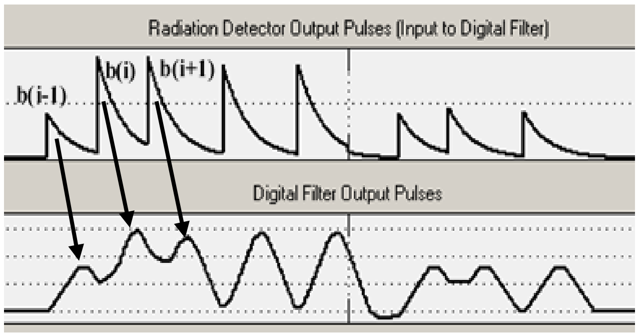


Figure 1: Pileup situation and response of a digital filter

The complete pulse train can thus be described by the timing of each pulse and the corresponding pulse amplitude, as well as a single model pulse shape common to all pulses that characterizes the dynamics of a single signal. Based on this model, the correlation of neighbouring signals can be resolved in order to restore the correct energy information from the measured amplitudes [3].

Hence, let us define *p(t)* to be the standard pulse shape. Then the *i*-th pulse with an amplitude a_i occurring at a time t_i can be modeled as:

$$pi(t) = a_i\, p(t - t_i) \tag{1}$$

The sample signal s(t) as a function of time t can then be described over time as a sequence of pulses:

$$s(t) = \sum_i a_i p(-t_i) \tag{2}$$

where both t and t_i are expressed in terms of the sampling period. However, the measured amplitudes are usually estimated by a weighted average of the original sampling sequence, in the vicinity of the peak of the pulse. This pileup recovery summation is constructed as a kind of finite response filter (FIR) [3], mathematically represented as:

$$q(t) = \sum_k f(k)s(t-k) \tag{3}$$

where *q(t)* is the response of such a filter, described by the weights *f(k)*. The presence of such a filter will induce a constant time lag l in the timing of the measured pulses with respect to their original time of occurrence. Ignoring this time lag, the amplitude b_i can be defined by [3]:

$$bi \ = \sum_j \left(\sum_k f(k)p\left(t_i - t_j - k\right)\right)a_j \tag{4}$$

As seen from Equation 4, the measured amplitudes are, in fact, just a linear combination of the real amplitudes. An example of pileup is illustrated in Figure 1. The amplitude of the pulse b_i is affected by the pulses b_{i-1} and b_{i+1}. In general, this can be expressed as [4]:

$$\mathbf{b} = \mathbf{Ma} \tag{5}$$

where the elements of the pileup recovery matrix **M** are defined as:

$$M_{i,j} = \left(t_i - t_j\right) \tag{6}$$

$$m(t) = \sum_k f(k)p(t-k) \tag{7}$$

The solution to Equation 5 is to invert the matrix **M** with the weights *Mi,j* to arrive at the true amplitudes a_i ie $\mathbf{a} = \mathbf{M}^{-1}\mathbf{b}$. A good approximate solution for the matrix is to assume small numbers of pulses and to solve the system of linear equations arising from them [4]. Assuming the simplest case in which the current pulse is affected by one pulse before and one pulse after, then in order to calculate the real amplitude of the current pulse, one needs to invert a 3 × 3 matrix. The elements of the matrix are assumed to be normalized. The matrix formed by the pulses *i − 1, i, i + 1* is described by [4]:

$$\begin{pmatrix}1 & p & 0\\ r & 1 & q\\ 0 & s & 1\end{pmatrix}\begin{pmatrix}a_{i-1}\\ a_i\\ a_{i+1}\end{pmatrix}=\begin{pmatrix}b_{i-1}\\ b_i\\ b_{i+1}\end{pmatrix} \quad (8)$$

where

$$p = m(t_{i-1} - t_i + 1),$$
$$r = m(t_i - t_{i-1} + 1),$$
$$q = m(t_i - t_{i+1} + 1),$$
$$s = m(t_{i+1} - t_i + 1) \quad (9)$$

From Equation 8, it follows that:

$b_{i-1} = 1 \text{ x } a_{i-1} + p \text{ x } a_i + 0 \text{ x } a_{i+1} = a_{i-1} + p \text{ x } a_i$

$b_i = r \text{ x } a_{i-1} + 1 \text{ x } a_i + q \text{ x } a_{i+1} = r \text{ x } a_{i-1} + a_i + q \text{ x } a_i$

$b_{i+1} = 0 \text{ x } a_{i-1} + s \text{ x } a_i + 1 \text{ x } a_{i+1} = s \text{ x } a_i + a_{i+1}$

(10)

For a digital filter, the pulse rise time (peaking time) is the same as the pulse fall time. Thus, due to the symmetrical digital filter output signal, the factors p, q, r and s of Equation 10 will be equal. Hence, if p = q = r = s = 1, then it will be observed that the outputs b_{i-1}, b_i and b_{i+1} have amplitudes greater than their corresponding inputs of a_{i-1}, a_i and a_{i+1}. To avoid any overflow of the output amplitudes with undetermined and undesirable consequences, the input needs to be scaled down by about 50%. This would ensure that the pulse shape and height are reasonably maintained within limits of the analogue-to-digital conversion (ADC) resolution subject to the time lag due to filtering. If we are interested only in the pulse heights, then we can equate the factors p = q = r = s = 0, in which case b_{i-1}, b_i and b_{i+1} will strictly correspond to the pulse heights of a_{i-1}, a_i and a_{i+1} respectively without considering their pulse shapes.

We present in Figure 2a, Equation 10 implemented in hardware form. If we apply the Nelder-Mead algorithm for peak search, we can compare the three values of b_{i-1}, b_i and b_{i+1} to finally obtain the recovered piled up pulses as b_i whenever the comparison criteria are met.

The analytical inverses for a_i, a_{i-1} and a_{i+1} are respectively equal to:

$$a_i = \frac{b_i - rb_{i-1} - qb_{i+1}}{1 - pr - qs}$$

$$a_{i-1} = b_{i-1} - \frac{p(b_i - rb_{i-1} - qb_{i+1})}{1 - pr - qs}$$

$$a_{i+1} = b_{i+1} - \frac{s(b_i - rb_{i-1} - qb_{i+1})}{1 - pr - qs} \quad (11)$$

The practical import of Equation 11 is that the denominator $(1 - pr - qs)$ should not be allowed to have a value of zero otherwise the matrix **M** becomes singular and a_i, a_{i-1} and a_{i+1} cannot be uniquely determined. Thus, the factors p, q, r and s should always have a value between zero and less than 1, making a_i to be an amplified form of its actual value. If we make p = q = r = s = 0.5, then b_{i-1}, b_i and b_{i+1} will have amplitudes with a maximum of 2 x a_i.

3 Design Methodology

We present below, the summary of the three major steps we followed to design the peak pileup detection and recovery units using Matlab and Xilinx blocks.

a. Step 1 – Design of random exponential pulse generator subsystem. The output of this subsystem simulates the input exponential signals from a pre-amplifier with varying arrival times and amplitudes.

b. Step 2 – Design of trapezoidal/triangular digital filter subsystem. The subsystem shapes the exponential input signals to trapezoidal/triangular pulses in a series of four stages.

c. Step 3 – Design of the pulse pileup detection, rejection and recovery units of the subsystem. This subsystem is made up of experimental units for normal pileup detection and rejection, hardware and software implementations of pileup detection and recovery. The units of the subsystem detect pulse peaks and pulse pileups and carry out the necessary recovery of piled up pulses either in a hardware or software manner.

3.1. Simulation of Radiation Detector Output Pulses – Random Pulse Generator Subsystem

The random pulse generator subsystem simulates the detector output pulses. In a real system, such signals are the outputs from the high pass filter and pole-zero cancellation circuits. The pulses have exponential shape with decay time equal to the noise corner time constant of the detector-preamplifier (usually in order of a few microseconds – 5uS). The random pulse generator is first implemented in a Simulink library and then added into the spectroscopy design. In this situation and from a customized library listed in Simulink library browser, we create a subsystem from the Simulink Library which we can label as **Random_Pulse_Generator** block. In addition, we included the following lines to the parameter script file to define the random exponential pulse characteristics:

```
% Clock Period
   Tclk = 0.02 e-6; % (50 MHz)
 % Pulse period
   Tpprd = 10e-6;
 % Clock period
   Tclk = 0.02*1e-6;
   Tclkn = 0.02*1e-6;
 % High pass filter constant
   Taud = 5e-6;
  % Sampling time
   TRSS = 5; %7.2; 8; 10; 12; 15
   T_st = TRSS*(1e-6);
  % Peaking time
   Taupk = T_st
   Taupk_top = 1.8*e-6
  % Random pulse generation sequences
   gg3=[0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
   1 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 0 1
   0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0
   0 0 1 0 0 1 0 0 0 0 0 0]';
  % Use TRSS = 5 - 7
   gg = gg3;
 % PHA group
   pha_x1 = 0.1;% Noise threshold
```

Other values stored in the parameter script file include the filter peaking time and top values, ADC resolution bits, pulse height analyser values and baseline restoration value. The simulated random exponential pulse outputs are as shown in Figure 1. It is to be noted that Steps 1 and 2 of the Design Methodology are not the subject of this writeup but are only needed for Step 3 to be realised.

3.2. Simulation of Peak Pileup Detection and Recovery Functions

The target of the peak pileup recovery algorithm is to find the relevant peaks in the spectroscopic raw data stream irrespective of pileup conditions as long as there is an identifiable peak in the stream. In this work, each three consecutive values are compared. To the best of our knowledge, there is no known open, custom-built or available commercial spectroscopy system based on the Xilinx Zynq-7000 SoC Board that can perform this function. However, we are aware of an effort such as in [5] in which the peak detector subsystem is implemented in a software mode through an Mcode block using a state machine to accept only the “clean” pulses while rejecting all other piled up pulses. The simple procedure employed here can be written by a pseudo-code as follows:

```
while (xn > dxn) do the following
{
If (xmax < xn) then xmax = xn
}.
```

In this case, the pulse peak detection and rejection function is performed by a subsystem which compares the current value with a temporary maximum ’xmax’. If the temporary maximum ’xmax’ is lower than the current value ’xn’, then the temporary value is replaced by the current value. This happens when xn < dxn, where ’dxn’ is a one clock interval delayed input signal ’xn’.

In this work in which the software design can be implemented with two circuit delay elements and simple codes in an Mcode block or in hardware form with a few simple logical operations using Matlab and Xilinx relational blocks, the peak is detected when the value in the middle is greater than the previous and the following. This approach can be implemented in any Xilinx FPGA based SoC such as the ZYNQ 7000 development board. The advantages of this algorithm are simplicity, low time consumption, easy to programme, and small number of logical elements or components involved. The subsystems in this design are implemented in both hardware form and software mode in which Nelder-Mead algorithm is also applied. Nelder-Mead algorithm is essentially a way of organizing and optimizing the changes in parameters to shorten the time required to fit a function to the required degree of accuracy. The most general way of fitting any model to a set of data is the iterative method. Consequently, iterative fit is performed as in the following general way for the peak pileup recovery based on [6]:

For three (3) consecutive peak samples X_n, X_{n-1} and X_{n+1} where X_n is the peak sample taken at the present time sampling instant, X_{n-1} is the peak sample taken one time sampling period before this instant and X_{n+1} is the peak sample taken during the next sampling instant, if $X_n < X_{n-1}$ and $X_n > X_{n+1}$ then X_n is a peak value. This algorithm can be implemented through a hardware form in an FPGA using the following steps:

a. The radiation signal is captured through the detector (This is simulated through the Matlab/Simulink workplace).

b. Implement appropriate stages of digital trapezoidal/triangular filtering, pole-zero cancellation and baseline restoration (also implemented through the Matlab/Simulink work place using Matlab and Xilinx blocks).

c. Implement the matrix linear equation mathematical relationships as provided in Equation 10 using appropriate Matlab and Xilinx blocks to provide outputs for b_{i-1}, b_i and b_{i+1}.

d. Provide two relational blocks which compare the three consecutive values of b_{i-1}, b_i and b_{i+1} together such that when 1st value (b_{i-1}) < 2nd value (b_i) >3rd value (b_{i+1}), a peak is detected in 2nd value (b_i).

e. The outputs of the two relational blocks of the previous step are connected to the inputs of an AND logical gate.

f. The output of the AND gate is considered as a selector of 2 to 1 multiplexer (MUX). If this output is 1 then this position is a peak and is selected by MUX otherwise the output of the MUX is zero. Thus, if MUX output is 1, then allow the corresponding Peak Values for further processing.

The software implementation of the algorithm is presented in Figure 2c.

4 Results and Discussion

With reference to the Design Methodology above, Steps 1 and 2 involve subsystems created to test the responses of the pulse peak pileup detection, inspection and recovery functions. Typical pulse shapes from the two subsystems are as shown in Figure 1.

To appreciate the results and the discussion thereof, it is important to compare the functioning of the Peak Detector of [5] and our designed generic Peak Detector and Recovery subsystems as shown in Figure 2b while the Matlab and Xilinx block diagram for the hardware pileup detection and recovery are as shown in Figure 2a. The code snippet for the software implementation is presented in Figure 2c. Their responses to an input exponential pulse train are shown in Figure 3a for a light pileup situation and in Figure 3b for a severe pileup situation. A severe pileup situation can be said to be a sampling situation in which the pulse peaks in a sampled pulse train are least identifiable after which they are no longer distinguishable.

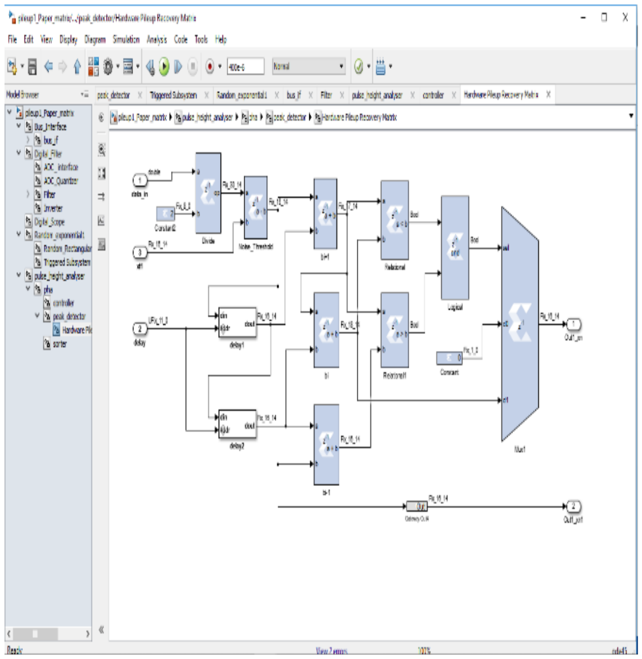


Fig. 2a: Matlab and Xilinx blocks for the hardware implementation of pileup recovery

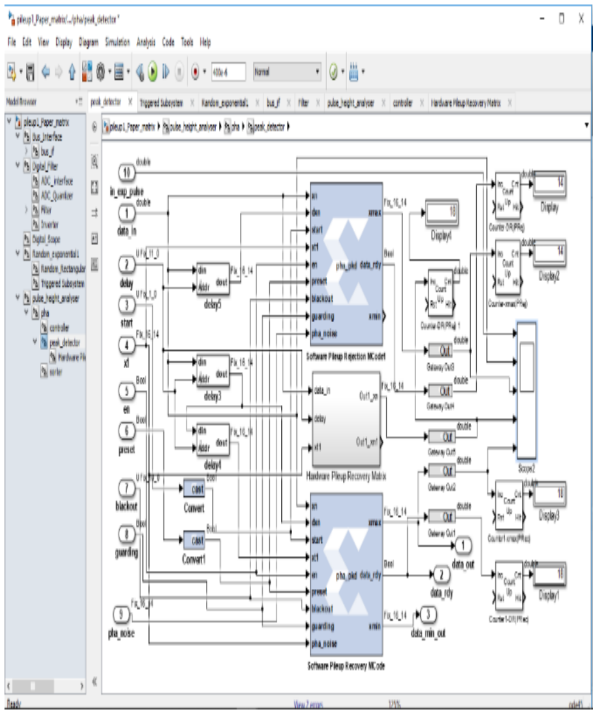


Fig. 2b: Simulation subsystems for pileup detection and rejection, hardware and software implementation of pileup detection and recovery

Code snippet for software implementation of pileup recovery

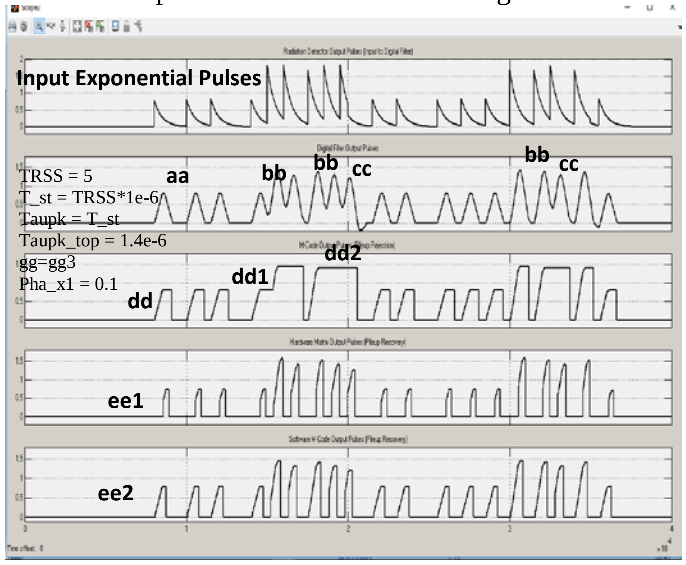
```
function [xmax, data_rdy, xmin] = pha_pkd(xn, dxn, start, xt1, en,
preset, blackout, pkd_counter_size, guarding, pha_noise)

if(state == s1)        %check threshold
        xmax_rdy = false;
        xmax1 = 0;
        tc = 0;
        tc_en = false;
        if (xn > pha_noise && xn > xt1 && xn > dxn && en == true)
%above threshold
           state = s2;
        else
           state = s1;        %below threshold: wait
        end
 elseif(state == s2)     %check if amplitude falls to 90%
        if (xmax1 < xn)     % not yet maximum: track for maximum
           xmax1 = xn;
        end
        if (xn < dxn && xn < xt1)        % pulse falls to X% of its amplitude,
X depends on delay
           if(xmax1 > xmin1 + pha_noise)
              xmax_rdy = true;
              tc_en = true;
           end
           state = s3;
        end
```

Fig. 2c: Code snippet for the software implementation of pileup recovery

Considering the simulation waveforms showing a light pileup situation (Figure 3a), the following observations could be made:

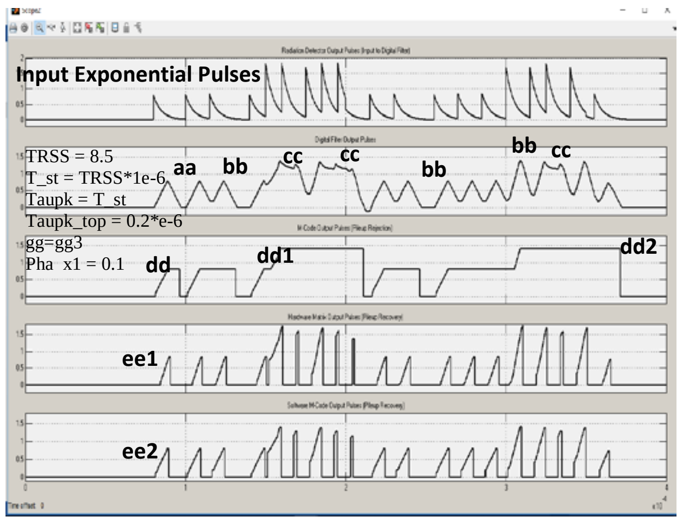
a. In the case of pileup rejection with the pulse width of a randomly generated exponential pulses of 10uS, the random pulse spacing with a minimum of 5uS and peaking time of the trapezoidal filter set at 5uS and the top at 1.4uS:

i. The pulses that are not affected by any other pulse within the duration of the trapezoidal/triangular pulse width (11.4uS) are allowed through as in (**aa**). Those pulses arriving after twice the peaking time and top (11.4uS) of an earlier trapezoidal/triangular filter pulses are also allowed through.



TRSS = 5
T_st = TRSS*1e-6
Taupk = T_st
Taupk_top = 1.4e-6
gg=gg3
Pha_x1 = 0.1

Fig. 3a: Pileup detection algorithms and output signals – Light pileup situation



TRSS = 8.5
T_st = TRSS*1e-6
Taupk = T_st
Taupk_top = 0.2*e-6
gg=gg3
Pha x1 = 0.1

Fig. 3b: Pileup detection algorithms and output signals – Severe pileup situation

ii. Those pulses which have peaked before the arrival of another pulse are also allowed through even when another pulse has started arriving as shown in (**bb**).

iii. Those pulses arriving before the elapsing of the pulse width of 11.4uS are rejected as shown in (**cc**). Thus, only fourteen (14) 'clean' pulses out of nineteen (19) incident pulses are allowed through to be recorded in the multi-channel analyser (MCA) as shown in (**dd**).The "clean" pulses include one pulse with two amplitude values as in **dd1** and **dd2** with the first amplitude **dd1** being the one that is binned in the MCA. Thus, altogether, 5 out of the 19 pulses are rejected giving a percentage rejection of about 26%.

b. In the case of pileup recovery as shown in Figure 3a, all the 19 peaks in the exponential pulse train are shaped in the digital filter with identifiable peaks. The whole 19 pulses are detected and recovered irrespective of their arrival times as shown in (**ee1)** for the hardware and (**ee2**) for the software implementation. It is observed that there is a constant time lag *l* in the time stamping of the measured pulses of the hardware implementation with respect to their original time of occurrence.

In the case of severe pileup situation shown in Figure 3b, all the 19 exponential pulses are shaped in the digital filter with identifiable peaks as shown in **aa**, **bb** and **cc**. It is observed that **cc** are severely piled up with barely identifiable peaks. For the pileup rejection, pulses around **bb** and **cc** are rejected. Altogether, 14 out of the 19 pulses are rejected giving a percentage rejection of about 74%. For both the hardware and software implementations, all the piled up pulses around **bb** and **cc** are 100% successfully recovered. The performances of both the hardware and software implementations are far better than in [3] where it was reported that more than 65% of the piled up pulses were recovered.

Furthermore, one can observe from the simulation that:

a. The dead time in this system is only limited to the pulse processing time given by the pulse width of the trapezoidal filter pulse. The pulse width varied from 1uS to 20uS, providing for 100% pileup recovery up to 17.2uS total trapezoidal pulse width. At a trapezoidal pulse width of 18uS, the hardware and software implementations had 89% and 84% pileup recovery respectively. At 20uS trapezoidal pulse width, they had about 79% pileup recovery each.

b. It is also observed that if the sum of 2 times the peaking time is not a multiple of the flat top time ie remainder of ((2 x Taupk)/Taupk_top) ≠ 0 and subject to a maximum of 17.2uS, then errors will occur. For example, at a peaking time of 8.4uS and a flat top value of 0.8uS, (trapezoidal pulse width of 17.6uS), some spurious detections, which could be attributed to noise jitters, may occasionally slip into the detection blocks especially for the hardware implementation as shown in Figure 4. For the noise to be effectively suppressed, the sum of 2 x peaking time and the trapezoidal flat top should be as much as possible a whole number such as 10uS, 11uS, 12uS etc.

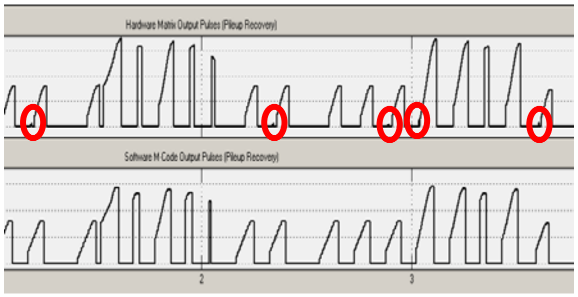


Fig. 4: Spurious detections by the detection blocks in the form of noise

c. For more practical applications, it is expected that the peaking time of the trapezoidal filter can be set to 0.8uS with a top of 0.2uS, providing for a maximum count rate without errors of over 500,000 counts per second, enough for most spectroscopic applications.

5 Conclusion

A linear equation matrix based mathematical algorithm for the detection and recovery of piled up pulses has been designed for implementation in an FPGA both in a hardware form using Matlab and Xilinx blocks and in a software format using a Xilinx Mcode block. This is achieved in conjunction with the Nelder-Mead algorithm which is essentially a way of organizing and optimizing the changes in parameters to shorten the time required to fit a function to a required degree of accuracy. The algorithm compares three consecutive values and detects a peak when the 1st value is less than the 2nd and the 2nd value is greater than the 3rd value. The 2nd value is the peak. Whereas in an earlier implementation based on an Mcode block using a state machine, out of a simulated pulse train of 19 pulses, only 5 pulse peaks were detected while the remaining 14 or about 74% were rejected in a severe pileup situation. In this present work, and for the same severe pileup situation, hardware and software implementations resulted in a 100% recovery of piled up pulses. Also, it has been shown that all the pulse peaks can be recovered from piled up pulses by using the derived mathematical matrix linear equation solutions and the Nelder-Mead algorithm to implement the hardware detection and recovery using Matlab and Xilinx blocks or by using two delay lines and writing appropriate codes in an MCode block for software implementation accordingly. Both approaches have better performances than a similar approach in an earlier work based on only the derived mathematical matrix linear equation solutions as reported in [3].

References:

[1] Mohamed S. El_Tokhy, et al, Comparative Analysis between Different Linear Filtering Algorithms of Gamma Ray Spectroscopy, Proc of the Intl Conf on Circuits, Systems, Signals, 2010.

[2] I.I. Mahmoud, M.S. El Tokhy, and H.A. Konber, Pileup Recovery Algorithms for Digital Gamma Ray Spectroscopy, IOP Publishing for Sissa Medialab, 2012.

[3] B. Loher, D. Savrana, E. Fioria, M. Miklavecd, N. Pietrallae, M. Venceljd. High count rate γ-ray spectroscopy with LaBr3:Ce scintillation detectors, Elsevier, 2012.

[4] Victor I. Stoica, Digital Pulse-Shape Analysis and Controls for Advanced Detector Systems, Printed by GVO Drukkers & Vormgevers B.V. Ponsen & Looijen, Groningen, 2012.

[5] M Bogovac, Digital Pulse Processor for Nuclear Spectroscopy, Nuclear Science and Instrumentation Laboratory, IAEA, 2015.

[6] Manar M Ouda, Mohamed S. El-Tokhy, Hardware Implementation for Pileup Correction Algorithms in Gamma Ray Spectroscopy, International Journal of Computer Applications, Vol 176, No 6, October 2017.