FPGA-Based Pulse Pileup Correction

M.D. Haselman[1], Member IEEE, S. Hauck[1], Senior Member IEEE, T.K. Lewellen[2], Fellow IEEE,
R.S. Miyaoka[2], Member IEEE,
[1] University of Washington Department of Electrical Engineering, Seattle, WA
[2] University of Washington Department of Radiology, Seattle, WA.

Abstract- **Modern Field Programmable Gate Arrays (FPGAs) are capable of performing complex discrete signal processing algorithms with clock rates above 100MHz. This combined with FPGA's low expense, ease of use, and selected dedicated hardware make them an ideal technology for a data acquisition system for a positron emission tomography (PET) scanner. The University of Washington is producing a high-resolution, small-animal PET scanner that utilizes FPGAs as the core of the front-end electronics. For this next generation scanner, functions that are typically performed in dedicated circuits, or offline, are being migrated to the FPGA. This will not only simplify the electronics, but the features of modern FPGAs can be utilizes to add significant signal processing power to produce higher resolution images. In this paper we report on an all-digital pulse pileup correction algorithm that is being developed for the FPGA. The pileup mitigation algorithm will allow the scanner to run at higher count rates without incurring large data losses due to the overlapping of scintillation signals. This correction technique utilizes a reference pulse to extract timing and energy information for most pileup events. Using pulses were acquired from a Zecotech Photonics MAPDN with an LFS-3 scintillator, we show that good timing and energy information can be achieved in the presence of pileup.**

I. Introduction

We are developing a second-generation data acquisition system to support several positron emission tomography (PET) designs being developed at the University of Washington [1]. It is based on our experience with the original MiCES electronics concepts [2]. Along with the development of the hardware, we are also developing algorithms for the field programmable gate array (FPGA) that will make up the core of the front-end electronics. In previous work, we have developed algorithms for statistical event location [3], digital timing [4], and automated pulse parameter discovery [5].

The main goal of this and previous work is to develop an all-digital FPGA-based signal processing suite for a small animal PET scanner. The addition of a pulse pileup correction routine will allow us to investigate experiments with higher count rates. This will be especially important for experiments that use a continuous scintillator crystal [6] or readout electronics with row-column summing. We are currently experimenting with scanner architectures that have both of these features. In [7] we show how a common anode timing channel for an MAPD array can be built. While this will lower the number of timing channels in the system, a common anode will increase the likelihood of pileup. The common anode is essentially a summation of all of the channels of a detector. So if two interactions occur anywhere within a detector in the timeframe of a single pulse, pileup will occur on the common anode.

This work will build upon our all-digital timing algorithm [4]. In this previous work, we utilize a high-resolution reference pulse that has the same shape as the scintillation pulses arriving from the ADC. In [5], we show how this reference pulse can be built in the FPGA out of the same pulses that it will be used on. This reference pulse is used to interpret the start time of the pulses. We also show how normalizing the pulse amplitudes by normalizing the area under the pulses can reduce time walk. The timestamp comes from a lookup table that stores the time elapsed from the start of the reference pulse based on the voltage of a sample.

II. Previous Work

In order to extract timing and energy information from a pileup event, the multiple pulses need to be separated. There have been previous analog and digital circuits implemented to recover from pulse pileup. The first analog method uses a dynamic integration technique to retrieve the energy information from a pileup event [8]. Dynamic integration uses an analog circuit to integrate a pulse until it detects a pileup event at which point it starts a new integration. To get the total energy value, the remaining tail of the first pulse is interpolated. The interpolated tail of the first pulse is subtracted from the second integration to resolve the energy of the second pulse. The high yield pileup event recovery (HYPER) method corrects for multiple pulse pileup events by computing a weighted sum of an overlapping pulse and subtracting the weighted sum of the previous pulses decreased by a time-decay term [9]. This method requires analog components and an integrator that clears when a pileup event occurs. This method has recently been converted into a digital implementation in an FPGA [10]. In order to achieve good energy resolution in a digital circuit, HYPER needs ADC rates of at least 200Msps as well as an analog trigger to signal the beginning of any pulse. Finally, after this idea was developed and the detailed implementation was studied, it was determined that a patent [11] exists that proposes a similar idea of pileup correction. This patent is targeted to gamma

Manuscript for NSS/MIC record received November 13, 2010. This work was supported in part by DOE grant DE-FG02-05ER15709Zecotech, Altera, and NIH grant EB002117.

Michael Haselman and Scott Hauck are with the Dept. of Electrical Engineering, University of Washington, Seattle, WA 98195 USA. (email: {haselman,hauck}@ee.washington.edu).

Thomas Lewellen and Robert Miyaoka are with the Dept. of Radiology, University of Washington, Seattle, WA 98195 USA. (email: {tkldog, rmiyaoka}@u.washington.edu).

cameras, so the ability to timestamp pulses was not proposed as we do in this work. The main concern of the patent was to obtain energy information from pileup events. So instead of actually separating the pulses, as this work does, only the energy contribution of individual pulses is calculated. As the following sections will show, in order to timestamp pulses involved in pileup, the pulses need to be separated.

III. Algorithm

In contrast to the methods discussed above, we aim to develop a pulse pileup correction algorithm on an all-digital platform that recovers the start time and energy of each individual pulse involved in tail pileup. Our proposed method uses a combination of pulse shape discrimination and partial dynamic integration to detect and remove peak pileup and to correct for tail pileup.

Fig. 1 shows the general structure of the algorithm for pileup correction. As a pulse is detected, the time stamp and pulse energy are calculated. After this, the pulse is removed from the data stream, which is sent to a second processing engine. By removing the first pulse, any pileup event present will be separated so that the second engine can process it for start time and energy. If no pileup is present, then the second engine will receive a flat signal. To remove the first pulse from the stream, the reference pulse is used to interpolate the tail of the first pulse that is hidden under any pileup event. The number of stages present in the algorithm is dependent on the amount of pileup events expected. For example, if the probability of having a pileup event that contains four pulses is less than 1%, then it isn't probably necessary to have more than three processing engines. The probability cutoff is a designer choice and is easy to change, as all of the internal stages (not first or last stage) are identical.

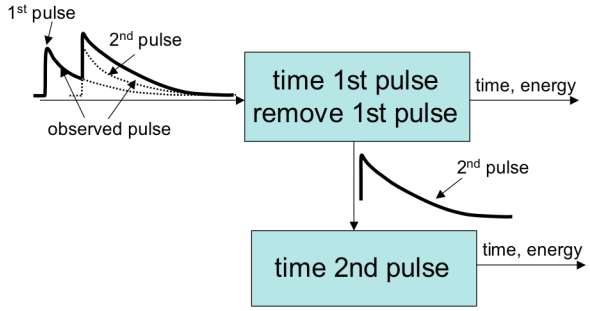


Fig. 1. Block diagram of the overall pulse pileup correction algorithm. Note that number of stages is dependent on the expected rate of pileup.

To determine the start time of the pulse, the timing algorithm [4] has to be modified, as the whole pulse can no longer be used to normalize that amplitude. Instead of using the whole pulse, only the area under the first portion of the pulse is used. The point that the summing is stopped will be designated as the transition from peak to tail pileup. This means that we will try to mitigate any pileup after this cutoff and remove any pileup before. The lookup table for area to amplitude normalization must be modified to reflect this change. In other words, when comparing the area of the data pulse to the reference pulse, the same number of samples is used in both summations. Instead of comparing the area of the whole pulse to the area of the whole reference pulse, the areas of the first portion of the two pulses are compared. The time lookup does not have to be modified. To eliminate pulses involved in peak pileup, the energy is also checked to determine whether it is below the expected energy maximum that was determined in the reference pulse discovery routine. The idea is that if a second pulse starts before the cutoff, then it will add energy to the summation. If it is determined that peak pileup has occurred, the pulses are discarded and the system is dead until the incoming data stream returns to baseline.

Once the first pulse is time stamped, it can be removed from the data stream. Because only tail pileup after the area summation cutoff will be corrected, only the samples after the cutoff need to be removed from the stream and the downstream engine only needs to process data after the cutoff for the above engine. This also means that the system can run in real-time, as no past data needs to be sent to the downstream engines. In this algorithm, it is assumed that pileup will occur, so the reference pulse is always used to remove the tail of the first pulse from the stream. The timestamp is used to determine what samples from the reference pulse to use. Recall that the reference pulse is defined at a much finer resolution (every 40ps in this work) than the sampling rate of the ADC. The data that is sent to the next processing engine is calculated using equation 1.

$$V[i] = V_{input}[j] - (V_{ref}[\left((n+i)\times\left(t_s/t_{ADC}\right)\right) + \Delta t]) \times \left(A_p/A_r\right) \quad (1)$$

Here, V_{input} is the data stream from the ADC, V_{ref} is the reference pulse voltage, n is the number of ADC samples summed on the leading edge of the pulse for the amplitude normalization, $\left(t_s/t_{ADC}\right)$ is the ratio of reference pulse resolution to ADC resolution, Δt is the result of the time stamp lookup table (i.e. how far the first sample was from the start of the pulse) and $\left(A_p/A_r\right)$ is the normalization factor to normalize the reference pulse amplitude to the first incoming pulse amplitude.

Partial dynamic integration is used to determine the energy of the pulse. Two summations are generated for each pulse, one to the cutoff point and one for the full pulse. If the downstream engine indicates that pileup did occur, then the cutoff summation energy is calculated by using the partial summation, with the remaining pulse energy calculated from the reference pulse as indicated above. The cutoff point was experimentally determine, which will be presented in the next section. If no pulse is detected in the downstream engine, then the whole pulse summation is used. This scheme requires a termination processing engine that simply indicates whether it has detected a pulse but does not process it. This engine would be in addition to the max number of expected consecutive pileups. Additionally, if this engine detects a pulse, the system is dead until the incoming stream from the ADC returns to baseline. The number of stages should be sufficient to handle the maximum number of consecutive pileup events expected. This will be a user setting based on the count rate, the logic available on the FPGA and the

acceptable level of data loss. Fortunately, the probability of seeing a pileup train of length n follows a Poisson distribution, so for reasonable count rates, the number of stages is low. For example, in our system with a pulse length of 200ns at 1Mcps, the probability of having a 4-pulse pileup is less than .5%.

IV. Tests and Results

A. Determining peak pileup cutoff

The first step in developing this algorithm is to determine where to set the cutoff between peak and tail pileup. We started by investigating how much of the leading edge of the pulse is needed to accurately calculate the area to amplitude normalization and interpret the tail of the pulse. The accuracy may degrade as less of the pulse is used, but using less of the pulse reduces the dead time, so the tradeoff between dead time and energy/timing resolution has to be balanced. To perform this, a simulation was performed in Matlab using 1000 pulses from different pixels on a Zecotech MAPDN with a LFS-3 scintillator.

To determine the effect on timing resolution, two streams were created with the pulses in each stream in coincidence. The normal timing algorithm was run, except only a portion of the pulse was summed and that area was used to normalized the amplitude. The amount that was summed was swept from 100% to about 5%. Fig. 2 shows the results of this study. A sample pulse is included to provide a reference of what part of the pulse is being sampled. The x-axis indicates how much of the pulse (in samples) is being summed up for amplitude normalization. For this sampling rate (65MHz) and pulse length, 32 samples constitutes summing up 100% of the pulse, while 2 samples corresponds to about 5% of the pulse (note it doesn't make sense to use just one sample). Notice that the timing resolution remains fairly flat until the cutoff point approaches the peak. If the summing is stopped before the peak, the timing resolution is substantially degraded, as would be expected.

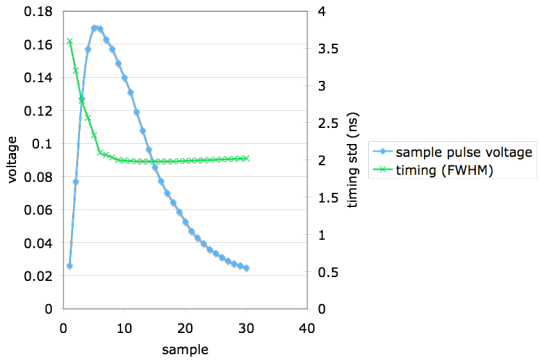


Fig. 2. Graph of timing resolution for different amount of pulse used for amplitude normalization for a 65MHz sampled pulse.

The next step is to determine how well the tail of the pulse can be interpolated based on summing up only a portion of the pulse. For this test, only one stream is needed. Again, for each percentage of pulse summed one thousand pulses were summed to the specified number of samples, while the rest of the pulse was interpolated with the reference pulse. The resulting composite summation is compared to the summation of the total pulse. The percent standard deviation of the error for all 1000 pulse is plotted in Fig. 3, along with an example pulse. The percent standard deviation demonstrates how much overall error is associated with interpolating the tail. The area error increases as less of the pulse is summed and more is interpolated. Again, there is a dramatic increase in error at about the peak of the pulse.

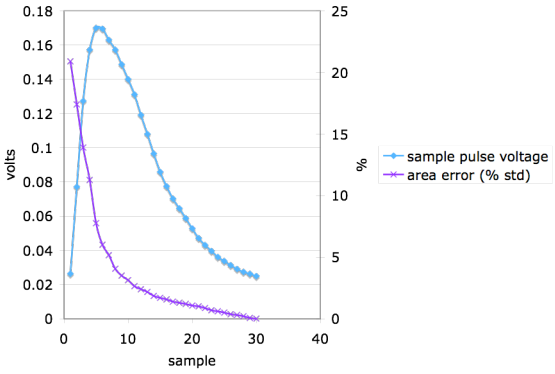


Fig. 3. Graph of area error as compared to how much of the pulse is interpolated for a 65MHz sampled pulse.

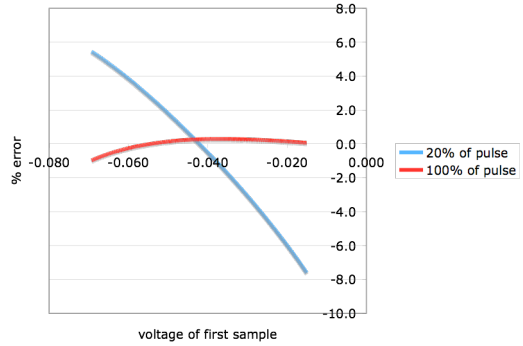
The results in Fig. 2 and Fig. 3 indicate that about 20% (7 samples for a 65MHz ADC) of the pulse is needed before the interpolation error becomes too great. In fact, for timing, the best results are obtained at about 30% of the pulse, and 20% is roughly equivalent to summing up the whole pulse. This is because the tail of the pulse contains a substantial amount of noise that corrupts the area normalization step. Also, the error in Fig. 3 will only be realized during a pileup event because otherwise the entire pulse is summed. Given these results, the line that distinguished peak pileup and tail pileup is set to 20% of the pulse. That is, if pileup occurs before 20% of the pulse, it should be detected and discarded. If it occurs afterwards, it can be separated with our algorithm. These results also indicate that there is not enough benefit to have a truly dynamic integration. That is, integrate until a pileup event is detected. To support this, a complicated trigger would be required to detect the pileup event and multiple normalization lookup tables would be required for each partial summation. It is possible to use a lookup table to determine the overall pulse area based on what percentage of the pulse is summed, as is done in [11], and use that in the normalization step, but this will add the to latency of each stage which will have consequences in a real-time FPGA implementation.

B. Area correction

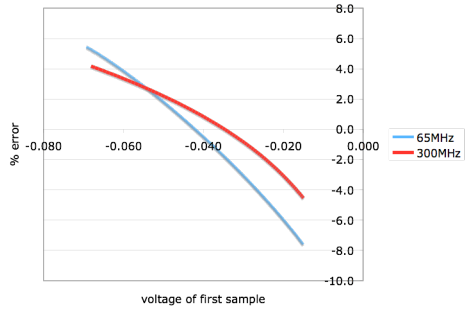
One issue that becomes a factor when just the first portion of a pulse is summed is the dependence of a pulse's calculated area on the voltage of the first sample. Because the pulse trigger looks for a sample above a set voltage, the range of possible voltages for the first sample ranges from the trigger to some greater value. The magnitude of this range is based on the slope of the leading edge of the pulse and the sampling interval. Of course, the greater the slope and the larger the sampling period, the greater the range. For the same pulse, a sampling that starts just after the threshold will have a lower calculated area than a sampling that starts well above the threshold.

Fig. 4 shows the extent of this error. Each graph shows the difference between the average area of a pulse and the area calculated for the same pulse for a given first sample voltage,

as indicated on the x-axis. The error is reported as a percent of the average pulse area for the amount of pulse summed. For example, Fig. 4a indicates that for a 65MHz ADC, the area summation is almost 8% under estimated when the first sample is just above the threshold (-.015V). There are no voltages below -.015V because that is the voltage level of the trigger, so no samples will be below this value. Fig. 4a demonstrates how the error is worse as less of the pulse is summed for the area to amplitude normalization. Notice how percent error for summing up all of the pulse is nearly flat and centers around zero, while the error for summing up only 20% of the pulse has large errors when the first sample is at either end of the range. Fig. 4b shows how this error trends for different sampling rates. A higher ADC means that there are more samples in the first 20% of the pulse and the range of the first sample is less. This results in less error than a lower sampling rate. However, the improvement from 65MHz to 300MHz is not large because the slope of the leading edge of the 300MHz-sampled pulse is greater because the frequency cutoff for the low-pass filter is higher. This is evident by the fact that the possible ranges of the first sample (x-axis) for both samplings is almost the same, though the range in time for the 300MHz sampling is ~1/5 that of the 65MHz sampling.



(a)



(b)

Fig. 4. Graphs indicating the area summation error based on the voltage of the first sample. (a) For a pulse sampled at 65MHz while sampling 20% of the pulse or the whole pulse. (b) For a pulse sampled at 65MHz and 300MHz summing up 20% of the pulse.

To correct for this, the error shown in Fig. 4 is calculated by sampling the reference pulse at all expected starting points and comparing the obtained area to the average area. This result can be stored in a look-up table that is indexed with the voltage of the first sample. After the initial area of 20% of the pulse is calculated, the first sample is normalized based on that area. The normalized voltage of the first sample is fed into the line equation, and the correction factor is calculated. Note that the line that is calculated is actual error and not percentage error as shown in Fig. 4, so the result of this calculation can be added to the area of the pulse. This corrected area is used to normalize the first samples of the pulse, which are used to determine the start time of the pulse.

The previous results, after correcting for the voltage of the first sample, are shown in Fig. 5 and Fig. 6. For both timing and area, there are no improvements when most of the pulse is summed which is logical considering how flat the 100% summation line is in Fig. 4a. The important difference in timing resolution and area error is when the summation limit approaches the peak of the pulse. For timing resolution in Fig. 5, correcting the area improves the timing at 20% summation by 6% and the standard deviation of the area interpretation reduces by 53%.

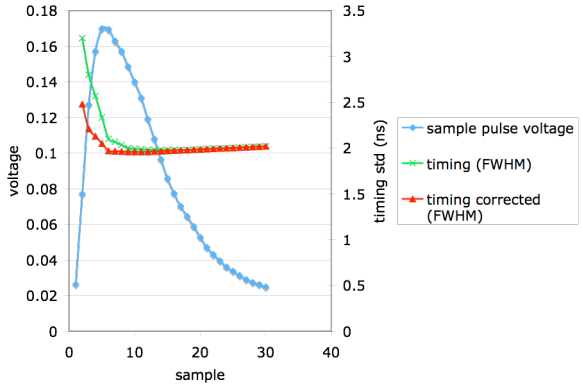


Fig. 5. Graph of timing resolution vs. amount of pulse summed with and without the area corrected for the amplitude of the first sample for a 65MHz sampled pulse.

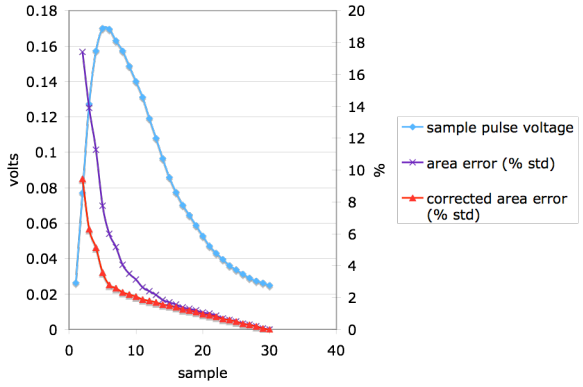


Fig. 6. Graph of area error with and without area correction as compared to how much of the pulse is interpolated for a 65MHz sampled pulse.

C. *Algorithm simulation*

From the previous tests, the algorithm is as follows:

1) remove baseline from incoming ADC stream
2) detect pulse based on triggers discussed [4]
3) sum up the first 20% and the whole pulse
4) check if first 20% summation is too large indicating a pileup event in the first 20% of the pulse
    a. if the energy is too great stall until the stream returns to baseline
    b. otherwise continue

5) normalize first sample of pulse based on area under first 20% of pulse
6) calculate the area correction factor and adjust initial area
7) normalize first samples (based on number of samples used in timing algorithm) of pulse to reference pulse using corrected area
8) timestamp pulse as discussed in [4]
9) normalize reference pulse to data pulse using corrected area
10) using timestamp, lookup the correct samples from the normalized reference pulse to interpolate remaining 80% of pulse
11) subtract interpolated pulse from ADC stream and send resulting stream to second pileup correction engine
12) if second engine detects a pulse it repeats steps 2-10 on the pulse and the first engine uses the energy based on 20% summed plus 80% interpolated
13) if no pulse is detected in the time it takes the first pulse to return to baseline, then the first engine uses the 100% summation for the pulse energy and it begins looking for pulses again on the ADC stream

To simulate the algorithm, 1000 pulses from four different pixels on a Zecotech MAPDN array with a LFS-3 scintillator were captured using a 25GHz oscilloscope. These pulses were then imported into Matlab to perform simulations. The pulses were captured as singles, so the coincidence and pileup had to be generated in Matlab before the above algorithm could be simulated. To facilitate this, two streams of pulses were created with one stream composed of pulses from a single pixel. To generate the streams, the first step is to randomly pick a start time for the pulse in stream one. Based on the desired percentage of coincidence, it is randomly decided if a coincidental pulse is placed in stream two at the identical start time. If it is decided that this pulse won't have a coincidental pair in the second stream, the start time for the pulse in stream two is also randomly selected. If however, it is determined that the current pulse will have a coincidental pair, then the start time in stream two is identical to the start time in stream one. Next, a pulse has to be chosen from the respective data sets to be placed at the determined start times. If these pulses are in coincidence, then the pulses are selected in order (i.e. the first coincident set are pulse one out of the thousand, the second set are pulse two and so on). This assures that all of the pulses from each pixel will be used in coincidence once and only once. If the pulses are not in coincidence, a random pulse is selected from the 1000 samples for the given pixel. Before the pulses can be added to the stream, the baseline has to be removed so that any overlap of pulses does not have multiple baseline components present. The baseline for each pulse is determined by averaging 300 samples just before the pulse. This value is subtracted from the pulse and then the pulse is added to the data stream. The array that makes up the data stream starts out filled with zeros, but as more pulses are added, the chance of two or more pulses overlapping increases. To generate a certain count rate, the length of the pulse stream is set such that after adding all of the coincidental pulses plus the random pulses, the number of pulses per unit time is correct. In this work we investigate 100 kilocount per second (kcps), 200kcps, 500kcps, and 1Mcps (below 100kcps, pileup is no longer a large issue, and 1Mcps is greater than the count rate we expect to handle). After all of the pulses are added to the stream, the baseline has to be added back between the pulses. To do this, baseline samples surrounding the pulses from the oscilloscope data are added to the zeros in the stream. When a pulse is intersected on the stream, the average of the previous 300 baseline samples added to the stream is added to the pulse. This adds a constant baseline to the stream without adding the baseline noise to the pulse, which would effectively double the noise. Note that where pulses overlap, the noise is doubled, as the noise cannot be removed from the pulses. The final step is to filter the stream and sample it with desired ADC sampling rate. The result of this routine is two streams that consist of randomly placed coincidental pairs with singles randomly placed in the stream. Fig. 7 shows a small section of the resulting streams for 1Mcps.

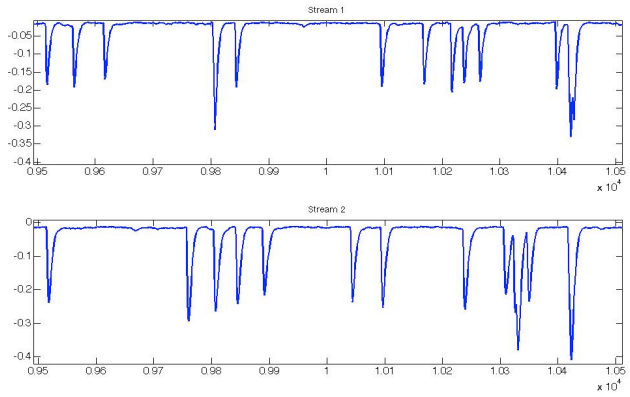


Fig. 7. Matlab plot of a pileup streams with 1000kcps from the pulse stream generator routine.

These streams were created with all possible unique pairings of the four pixels (six pairings), for five ADC sampling rates (65MHz, 125MHz, 300MHz, 500MHz, 1GHz), and with four count rates (100kcsp, 200kcps, 500kcps, 1Mcps). The energy resolution and coincidence timing resolution was recorded for each test. The timing resolution was averaged over the six pixel pairings. The results for how timing resolution is affected by the count rate are shown in Fig. 8. As expected, the timing resolution improves as the ADC rates increase, and it degrades as the count rate increases because more pulses are involved in pileup. The timing resolution from 100kcps to 1Mcps degrades by about 17% for a 65MHz-sampling rate and about 38% for the other sampling rates. Fig. 8 also shows that the degradation is essentially linear over the sampling rates covered. This indicates that our algorithm degrades gracefully as pileup rates increase.

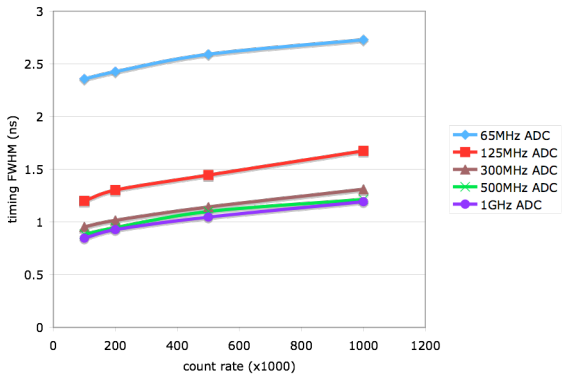


Fig. 8. Timing resolution for different count rates at different ADC sampling rates.

To get an understanding of what is contributing to the degradation, the timing resolution was calculated for each possible coincidence scenario. That is, coincidental pulses when neither pulse were involved in pileup, when one of the two were in a pileup event, and when both pulses were some part of a pileup event. These results are presented in Fig. 9 for a 300MHz ADC sampling rate along with the overall timing resolution. Note that data for 100kcps for one pulse involved in overlap and the data for 100kcps and 200kcps for both involved in pileup are missing. This is because there were not enough instances of those events at low count rates to make the data statistically sound.

From this data, it appears that most of the degradation is from the pileup events, and especially from the case when both pulses in coincidence are involved in pileup. There is still some degradation in the timing resolution for pulses not involved in pileup when count rates increase. The degradation is about 17% from 100kcps to 1Mcps. This is probably due to the effect of the ability to calculate a good baseline when pulses are close together but not overlapped. Also, as will be discussed in the next section, not all of the pulses classified as "no overlap" may actually have pileup.

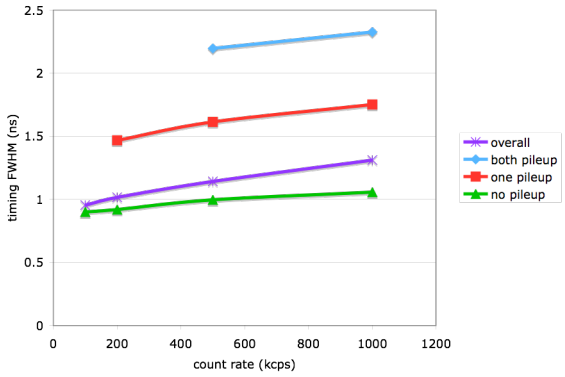


Fig. 9. Plot of timing resolution for a 300MHz ADC sampling for different count rates. The timing resolution is shown for coincidental pulses where neither of the pulse is in a pileup event, where one of the pulses is piled up and where both of the pulses had to be separated from another pulse due to pileup. The overall timing resolution is the combination of all three.

In addition to timing resolution, energy resolution is an important factor of a PET pulse processing system. For the pileup correction algorithm, the energy resolution will give an indication of whether the energy of the pulses involved in pileup can be accurately estimated. To calculate the energy resolution, the energy of all pulses in the stream (except those with too much pileup) was recorded and a histogram was generated. The data set used in these experiments only contains photoelectric events, so only the photo peak appears in the histogram. To determine the FWHM, a Gaussian function was fit to the data and the energy resolution was calculated. The energy resolution results are reported in Fig. 10. Notice that the energy resolution is fairly constant over different count rates. In fact, the worst degradation from 100kcps to 1Mcps is less than 5%.

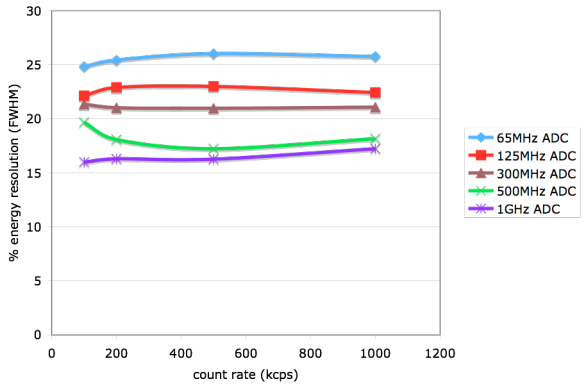


Fig. 10. Graph of the energy resolution of our pulse pileup correction as count rates increase for different ADC sampling rates

Like was done for the timing resolution, the energy resolution can be broken down into the contribution of different kinds of pulses. There are four different pulse classifications for energy resolution.

1) no pileup – pulse not involved in pileup
2) tail interpolate – first pulses in a pileup event where the tail of the pulse had to be calculated with the reference pulse
3) pileup – pulses that are last in a pileup event where the previous pulse had to be subtracted out
4) pileup with tail interpolate – pulses that are in the middle of a pileup event where the previous pulse had to be subtracted out and the tail had to interpolated from the reference pulse

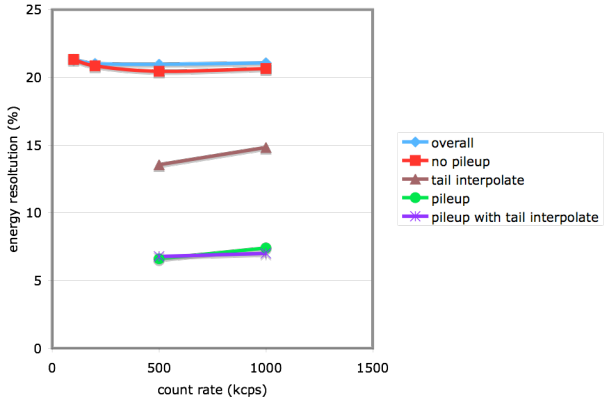


Fig. 11. Energy resolution for a 300MHz ADC broken down to pulses in different places of a pileup event.

Again, there are not enough pileup events at 100kcps and 200kcps for any reliable results for the tail interpolate, pileup or pileup with tail interpolate classifications. The most interesting result from this simulation is how much better the

energy resolution is when some of the pulse has to be interpolated using the reference pulse. This is because the reference pulse doesn't have nearly as much noise as the individual data pulses. This is also reflected by the result that the overall energy resolution and the energy resolution with no pileup are almost identical. Recall that when there is no pileup, that the pulse is summed up to 100% for the energy calculation.

D. Pulse detection

Since the purpose of pulse pileup correction is to capture additional pulses, it is important to investigate pulse detection efficiency. That is, how well does the algorithm detect good pulses and reject peak pileup pulses that have too much pileup. To determine how well our algorithm is detecting pulses, statistics of pulse detection were kept as they were discovered and processed. Pulses were classified by the number of pulses in a given pileup event. For example, 2-pulse pileups are events where two pulses piled up in a row, and 3-pulse pileups are events with three pulses in one pileup. The number of pulses without pileup, as well as pulses with too much pileup (peak pileup), was also recorded.

Analyzing our pileup correction algorithm indicates that it is a paralyzable system. Recall that if pileup occurs in the first 20% of a pulse, then the system is dead until the incoming ADC values return to baseline. So, if a third pulse arrives before the system returns to baseline, then the dead time is extended by another pulse length. So our system has essentially two dead times. The dead time when the system is live (not resolving peak pileup) is 20% of the length of a pulse. That is, when the system is live, the minimum separation required between two pulses is 20% of the pulse length. When peak pileup does occur, the dead time is then the full pulse length. The system is dead for at least one pulse length until the data stream returns to baseline. Given these parameters, the pileup rates cannot be calculated with the typical statistical models that assume a single dead time, so instead the pileup rates were calculated with a Monte Carlo simulation. Specifically, the rates were tabulated from the stream generation routine. After each stream was generated, the start times of every pulse were evaluated to determine how many pulse "groups" had no pileup, and how many were a 2-pileup, 3-pileup or 4-pileup event. The occurrence of peak pileup was also determined. The results in Table I show the percent difference from detected to expected for each subcategory.

Table I. Percent differences from detected pileup events to expected pileup events (detected count/expected count).

count rate	no pileup	2-pulse pileup	3-pulse pileup	4-pulse pileup	peak pileup
500kcsp	0.77% (1691/1678)	5.4% (129/122)	33.3% (6/4)	0% (0/0)	-175% (12/33)
1Mcps	0.68% (1464/1454)	13.7% (190/164)	13.6% (22/19)	0% (5/5)	-137% (27/64)

For these tests (Table I), the algorithm detects less than half of the events when pileup occurs in the critical region. The rest are classified as a normal pulse. This is the reason that the algorithm detects more pulses with no pileup, 2-pulse and 3-pulse pileup events. This indicates that our mechanism for detecting pileup in the first 20% of the pulse is not robust enough, and some of the pulses that should be classified as too much pileup are making it through the filter.

To test this hypothesis an experiment was set up to determine the ability of our algorithm to detect pileup before the critical point. Recall that the test for too much pileup is checking for too much energy in the first 20% of the pulse. A simulation was generated where 200 different pulses from the same pixel were put into a stream so that 100 2-pulse pileup events were generated. The degree of pileup for all 100 events was the same for each test. The pileup started at 1% of the first pulse (essentially two pulses directly overlapping) and was swept to 20% (the peak pileup cutoff point). In other words, about 20 tests were run with all of the 100 pileup events having the same amount of pileup. For each test, the peak pileup detection scheme was run and the number of pulses that made it through the filter (did not detect peak pileup) was noted. Of course, all pulses should be detected as peak pileup because they are all before the peak pileup cutoff. The results are tabulated in Fig. 12. For any pileup in the first 7% of the pulse, our peak detection scheme will accurately classify all pulses as having too much pileup to resolve. From about 7-15% the number of pulses that pass through the peak pileup detection filter increase until about 16%, where all of the pulses are incorrectly classified as not having pileup before the critical point cutoff. The reason that Fig. 12 reaches 100% missed at 16% overlap is because of the margin built into the max pulse energy for second (or third) pulses of a pileup event. The area of the second or third pulses in a pileup event contains the error associated with removing the interpolated tail of the previous pulse. This error can cause the area to increase (or decrease); therefore a small margin has to be included in the max pulse energy to eliminate the possibility of filtering out good pulses. Considering this, it is possible to shift the curve in Fig. 12 to the right at the potential cost of missing some good pulses.

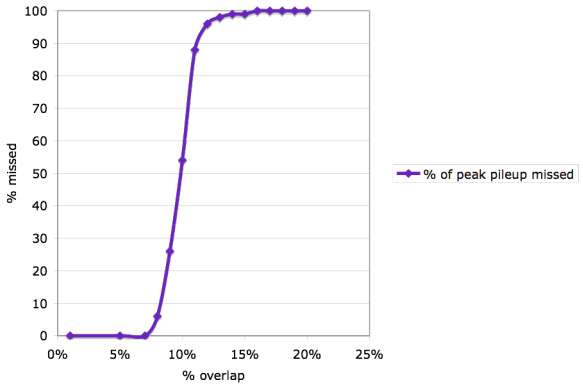


Fig. 12. Plot of the percentage of pulses that incorrectly determined to not have peak pileup for pileup events in the first 1-20% of the pulse.

The reason for this inefficiency of the peak pileup detection routine lies in the energy parameters of the pulses. Unfortunately, the range of energy of a pulse without pileup can vary by almost 2X in this data set. This is for pulses from a single pixel in a MAPD array. This range may be even larger for the common anode design, as it will be a signal derived from all pixels in the array. This means that if the first pulse is on the lower end of the energy spectrum, then more energy from a pileup event (i.e. greater overlap) is

required to trigger the filter. Therefore, this pileup event will not be filtered as peak pileup.

The next question from these results is what is the effect of these misclassified pulses on the timing and energy resolution of our algorithm? To determine the extent of this error, the same streams were generated as discussed above, but without any pileup in the first 20% of any pulse. This will remove the need for the peak pileup filter, as no peak pileup will exist in the data set. Theses streams were evaluated with our algorithm in the same manner discussed above. The results of this test will indicate how much of a determent the missed peak pileup events have on the system. In essence, we determined how much of the increase in timing resolution from 100kcps to 1Mcps in Fig. 8 is due to these errors. The results of this test for a 300MHz sampling ADC are shown in Fig. 13.

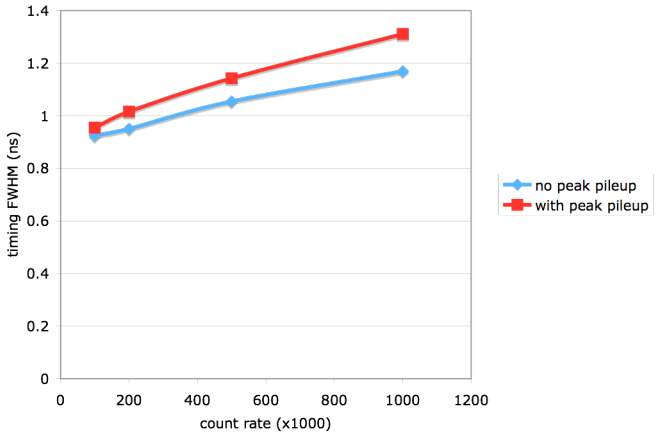


Fig. 13. Timing resolution versus count rate for a 300MHz sampling ADC. The top line is for a test that contains peak pileup, while the lower line shows the timing resolution for a test with peak pileup removed.

The elimination of all peak pileups from the processed data set improves the timing resolution by 11% at 1Mcps. This represents the ability of our algorithm if the peak pileup filter was perfect.

When peak pileup is removed from the stream, the energy resolution is about 21% for all of the count rates. This indicates the increase in energy resolution is because of the peak-pileup that is erroneously included into the data set.

While the energy resolution is not greatly affected by the inefficiencies of the peak pileup filter, there are some applications where an accurate measure of the energy of each individual pulse is necessary. In these scenarios, any addition energy from the second pulse would adversely affect the algorithms that use the pulse energy. One possible solution in an application that requires accurate pulse energy is to check for too much energy in the pulse using a summation to 35% of the pulse. The amplitude normalization would still occur with 20% of the pulse. This would presumably move the curve in Fig. 12 to 22-30%, so the peak pileup filter would eliminate all pileup before 22% of the pulse. Of course, this would increase the theoretical dead time of the system to 35% of the pulse, which would reduce the number of pileup events that are mitigated.

Another possible implementation to detect peak pileup is to use a two-part filter; one that looks for a second rising edge as well as too much energy in the first 20% of the pulse. Since the peaks of the pulses are generally in the same location (~10% of the pulse), any drastic increase above the previous sample after the peak could be classified as a peak pileup event. This would detect many of the peak pileup events that occur from the peak of the pulse to the peak pileup cutoff. The existing energy filter would detect any pileup before the peak. These possible peak-pileup filters will be investigated in future work.

V. Discussion

In this work, we show that an all-digital pulse pileup correction algorithm can reliably recover pulses overlapped up to 80%. In fact pulses with slightly greater overlap are still processed (because they don't have greatly increased integration values) albeit with a slight degradation in energy and timing resolution. This algorithm ties in very well with our timing algorithm [4] and can easily be implemented in an FPGA. The only complication in an FPGA implementation is the latency between the first processing engine and the last. This is because the downstream engines communicate with the first engine to indicate that it is free to look for the next pulse. If there is a large latency between samples coming from the ADC and samples going into the last engine, then there is the possibility of missing some ADC samples.

References

[1] T.K. Lewellen *et al*., "Design of a Second Generation FireWire Based Data Acquisition System for Small Animal PET Scanners," *IEEE Nuclear Science Symp. Conf. Record*, 2008, pp *(NSS/MIC)*. 5023-5028.

[2] T.K. Lewellen, M. Janes, R.S. Miyaoka, S.B. Gillespie, B. Park, K.S. Lee, P. Kinahan: "System integration of the MiCES small animal PET scanner", *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC)*, 2004, pp. 3316-3320.

[3] DeWitt D, Miyaoka RS, Li X, Lockhart C, Lewellen TK., "Design of a FPGA Based Algorithm for Real-Time Solutions of Statistics-Based Positioning," *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC)*, 2008, pp. 5029-5035.

[4] M.D. Haselman, S. Hauck, T.K. Lewellen, and R.S. Miyaoka, "Simulation of Algorithms for Pulse Timing in FPGAs," *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC)*, 2007, pp. 3161-3165.

[5] M. Haselman, S. Hauck, T.K. Lewellen., R.S. Miyaoka., "FPGA-Based Pulse Parameter Discovery for Positron Emission Tomography," *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC),* 2009, pp. 2956-2961.

[6] R.S. Miyaoka, Xiaoli Li, C. Lockhart, T.K. Lewellen, "New continuous miniature crystal element (cMiCE) detector geometries", *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC)*, 2009, pp. 3639-3642.

[7] Y.C. Shih et. al.,"An 8x8 row-column summing readout electronics for preclinical positron emission tomography scanners," *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC),* 2009, pp. 2376-2380.

[8] Lewellen TK, Bice AN, Pollard KR, Zhu JB, Plunkett ME, "Evaluation of a clinical scintillation camera with pulse tail extrapolation electronics", *J. Nuclear Medicine,* 1989, vol. 30. pp. 1544 –1558.

[9] Wong, W.H., Li, H., "A Scintillation Detector Signal Processing Technique with Active Pileup Prevention for Extending Scintillation Count Rates," *IEEE Trans. Nuclear Medicine*, vol. 45, no. 3, pp. 838-842.

[10] Liu, J. et al., "Real Time Digital Implementation of the High-Yield-Pileup-Event-Recovery (HYPER) Method," *2007 IEEE Nuclear Science Symposium Conference Record,* M26-4, pp. 4230-4232.

[11] R.E. Arseneau, "Method and Apparatus for Unpiling Pulses Generated by Piled-up Scintillation Events," U.S. Patent 5 210 423, May 11, 1993.