FPGA-Based Pulse Parameter Discovery for Positron Emission Tomography.

M. Haselman, Member IEEE, S. Hauck, Senior Member IEEE, T.K. Lewellen, Fellow IEEE,
R.S. Miyaoka, Member IEEE,

***Abstract*-Modern Field Programmable Gate Arrays (FPGAs) are capable of performing complex digital signal processing algorithms with clock rates well above 100MHz. This, combined with FPGA's low expense and ease of use make them an ideal technology for a data acquisition system for a positron emission tomography (PET) scanner. The University of Washington is producing a series of high-resolution, small-animal PET scanners that utilize FPGAs as the core of the front-end electronics. For these next generation scanners, 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 how we utilize the reconfigurable property of an FPGA to self-calibrate itself to determine pulse parameters necessary for some of the pulse processing steps. Specifically, we show how the FPGA can generate a reference pulse based on actual pulse data instead of a model. We also report how other properties of the photodetector pulse (baseline, pulse length, average pulse energy and event triggers) can be determined automatically by the FPGA.**

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] and coincidence timing [4].

One goal of our previous work was to produce an all-digital FPGA-based timing algorithm that could achieve a resolution that is suitable for small-animal scanners with current serial ADC sampling rates (<100MHz). The other design criteria were that timing resolution should scale with technology improvements of ADCs and that the algorithm should eventually outperform the analog version, given a fast enough sampling period for the ADC. The results from that study [4] reported that the two goals were met, but the algorithm relies on information about the system that may difficult to obtain, may change over time and/or vary between photodetectors. Specifically, our algorithm utilizes a reference pulse with the same shape as the sampled pulses from the photodetector.

Manuscript for NSS/MIC record received November 20, 2009. This work was supported in part by Zecotech, 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).

The algorithm also utilizes the pulse length, baseline restoration, and event triggers that are very close to the baseline. Our research has indicated that some parameters can vary between photodetectors (reference pulse shape and pulse length) and over time (event triggers and baseline). In this work, we have developed tools and algorithms for the FPGA to automatically determine each of these parameters. This will allow the FPGA to tune itself to each sensor, as well as adjust to the system as it changes over time.

One complication of our timing algorithm, as well as any that use a reference pulse [5], is the need to accurately determine or discover a good reference pulse. Previous timing work [5] used exponentials as models for the photodetector pulse. Our experiments found that models based on exponentials did not fit the data as well as models derived from the actual photodetector pulses. One reason for this is the inability to accurately model the electronics between the photodetector and the FPGA. All components of this chain (amplifier, low-pass filter, ADC, FPGA input pads, printed circuit board) affect the shape of the final pulse. Additionally, we found that for MAPDs, each pixel can produce a different shaped pulse. All of these factors indicate that a solution where the FPGA builds reference pulses using sampled data may produce the best possible timing results.

To exploit this, we have developed an algorithm that utilizes amplitude normalization and a timing lookup technique to build and refine a reference pulse. The process is to reconfigure the FPGA to capture and store many event pulses that are sampled at the ADC period (~15ns). These pulses will then be used to form the reference pulse that is defined at a much finer time step (60ps).

II. FPGA Timing

The data acquisition electronics that we are developing will have an Altera StratixIII S200 FPGA as the main processing component. A single FPGA will process all 64-detector channels supported by each board. Processing this many channels on a single board prohibits the use of a discrete timing circuit for each channel. Another implication of this architecture is that serial ADC's must be used in order to meet the I/O constraints of the FPGA. This currently limits the sampling period of the ADC to around 100MHz. This has lead us to develop an all digital timing technique that can produce high resolution coincidence timing based only on the sampled photodetector pulse.

The timing technique that we developed is a leading edge detector with pulse amplitude normalization. Using the leading edge is preferable because it is composed of photons that travel directly to the photodetector while the tail of the pulse is composed of photons that have reflected in the scintillator before reaching the photodetector. This means that the leading edge will be much more consistent (ignoring depth of photon interaction). One issue with leading edge discriminators is the phenomenon known as time walk. This refers to the time difference for pulses of differing amplitudes to reach a given voltage as shown in Fig. 1. In other words, a pulse with lower amplitude will take longer to reach a threshold voltage than a pulse with higher amplitude. This time differences will introduce jitter into the coincidence timing if it is not corrected.

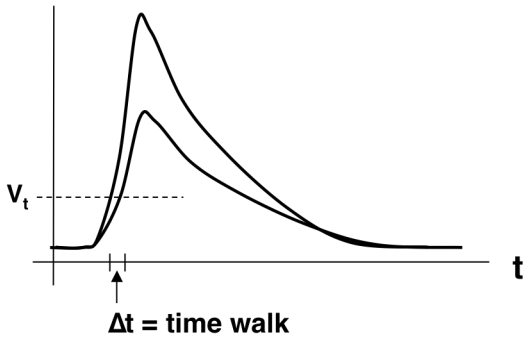


Fig. 1. Illustration of the timing jitter introduced by time walk when two pulsed have varying amplitudes.

We have developed an FPGA timing technique motivated by these two concepts. First, in order to get the most accurate timing information, the threshold to determine the start of a pulse is put as close to the baseline as possible without being triggered by noise. Once a pulse is detected, the amplitude is normalized to a preset value. This is accomplished by normalizing the area under the pulse or the summation of all of the samples that compose the pulse. This works because all of the pulses from a single photodetector have similar exponential shape with varying amplitudes so normalizing the area (integral of exponential) also normalizes the amplitude. After normalization, the data from the leading edge is used for determining the start time of the pulse. Previous research [4] indicated that the best timing resolution was achieved when just the first sample above the threshold was used for timing. A lookup table is used to convert the voltage of this first sample to a pulse start time. The input to the lookup table is the voltage of the sample, and the output is the time is takes for the reference pulse to reach that voltage on the leading edge. The reference pulse is a pre-computed curve that has the same shape as the incoming photodetector pulses but is defined at a much higher resolution (every 60ps in this work). The reference pulse also has the same amplitude that the incoming pulses are normalized to. The output of this lookup table is subtracted from the course grain time (counter based on ADC rate) to produce the timestamp. The course grain time is the count of the first sample above the threshold.

To determine the resolution of this timing algorithm, pulses were acquired from a Zecotech Photonics MAPDN with an LFS-3 scintillator using a 25GSPS oscilloscope. The pulses were imported into Matlab and subsampled to the ADC rate of interest. The coincidental timing for a 65MHz subsampling is 2.5ns. With a 125MHz subsampling, the timing improves to 1.4ns. This simulation indicates that the timing resolution of this algorithm is suitable for our small animal PET scanner.

However, when this algorithm was implemented in an FPGA, the timing resolution was more than 2X worse [7]. This large discrepancy is mostly due to the inability to calculate an accurate reference pulse for the voltage to time lookup table and the amplitude normalization function. For the implementation experiment, photodetector pulses were collected with a 25GSPS oscilloscope connected between the low-pass filter and the ADC in Fig. 2 (closest possible connection to the FPGA without modifying FPGA development board). These pulses were then imported into Matlab, averaged to determine a reference pulse and then imported into the FPGA. The main problem with this approach is that the affect the ADC, FPGA I/O and the printed circuit board have on the shape of the photodetector pulses is not captured when pulses are sampled before the ADC. An alternative method to determine the reference pulse is mathematically deriving the reference pulse. Previous work [5] has modeled the pulses as two exponentials, but our research indicated that while exponentials are a close fit, they are not accurate enough for our needs. Moreover, the inaccuracy is largest at the initial portion of the pulse where the most accurate timing information is. Furthermore, if a mathematical function is used, an accurate model of the data acquisition chain in Fig. 2 must be calculated in order to determine how it will influence the shape of the pulse that reaches the FPGA core. This model is difficult to get correct and would have to be performed for each photodetector of a scanner.

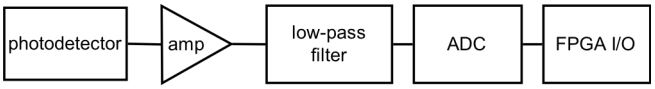


Fig. 2. Typical data acquisition chain for a PET scanner.

III. Reference Pulse Discovery

Alternatively, the FPGA can form a reference pulse from the sampled pulses. To do this, the FPGA must be able to build an accurate reference pulse defined at a higher resolution (every 60ps in this work) from pulses that are sampled at the ADC rate (~15ns). This method has multiple advantages. The first is that the acquisition chain in Fig. 2 no longer needs to be modeled, as any affect the chain has on the shape of the final pulse is already present in the pulses the FPGA processes. The other main advantage is that this automates the discovery of the reference pulse as well as other pulse parameters. This allows for each channel to have it's own reference pulse without going through the tedious process of acquiring pulses from each channel with an oscilloscope and generating the reference pulse offline. Using the oscilloscope approach for a whole scanner would be very impractical. Furthermore, if any components in the acquisition chain are upgraded (e.g. different low-pass filter, faster ADC) a new reference pulse has to be calculated. The automation of the FPGA reference pulse discovery allows for easy investigation of optimum filtering, new detectors or any other changes in the acquisition chain.

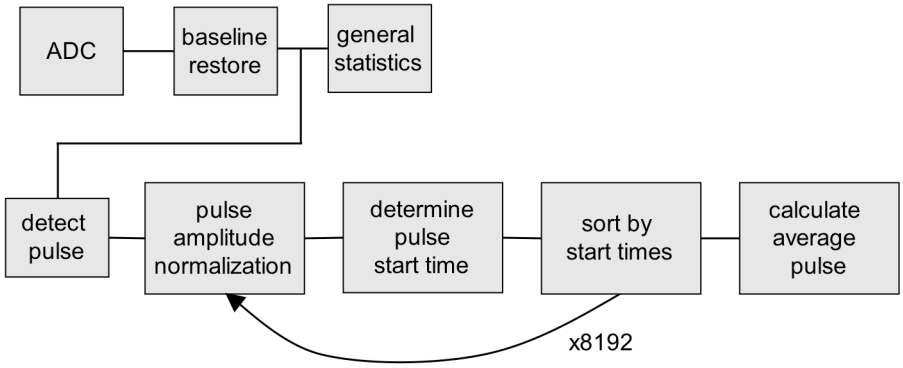


Fig. 3. Block diagram of the overall reference pulse discovery algorithm.

The overall algorithm is shown in Fig. 3. As samples arrive from the ADC, the first thing that is calculated is the baseline value. Solid-state photodetectors have a non-zero baseline voltage that is present in the absence of any photon interactions. With the baseline removed, the following general statistics are determined in the order as shown.

1) event trigger threshold voltage
2) average pulse length
3) average pulse energy

After these statistics are calculated, pulses can be detected and processes. Pulses are detected by looking for four contiguous samples over the event trigger threshold voltage. Once a pulse is detected, the amplitude is normalized so that all of the pulses utilized to make the reference pulse have the same amplitude. The next step is to determine the start time of the pulse in order to line them up in time. The start time is then used to sort the pulses into the higher resolution array that is used in the final step to calculate the reference pulse. The final step of calculating the reference pulse is done after 8192 pulses are collected.

A. *Baseline Restore*

An issue seen in systems that utilize solid-state photodetectors is a non-zero baseline for event pulses due to the dark currents. This baseline can shift over time, so to calculate the most accurate energy and timing values the baseline should be calculated constantly during runtime. Unfortunately, there is too much noise in the baseline to use a single point just before the pulse. Additionally, there is no guarantee that the points before a pulse are not a tail of another pulse, so averaging a small number of samples before the pulse may give incorrect results as well. Given these constraints, any baseline restore algorithm should fulfill these two requirements:

1) able to react to baseline moves but not noise
2) can not be affected by pulses

We have developed a solution that accurately calculates the baseline and adheres to these requirements.

As each sample is read into the FPGA, the first step is to determine if the sample is a part of an event or a sample of the baseline. If the sample is a part of an event pulse, it should not be included in the baseline calculation. To exclude samples of an event pulse, a baseline window is calculated. If a sample is inside the baseline window, it is determined to be a part of the baseline otherwise it is assumed to be a part of an event pulse. The requirements for the baseline window are as follows:

1) must include most values of the baseline including noise
2) must be able to track baseline shifts
3) must be able to expand or contract based on noise levels

All of these requirements indicate that the baseline window cannot be static but must be capable of expanding and contracting. The window also needs to be two sided in order to eliminate large noise spikes in the opposite direction of the event pulses. To meet these requirements, the baseline window keeps track of how many samples have been inside of the window and how many have been outside of the window. If 512 samples have been inside of the window, then the window contracts by one ADC least significant bit (lsb). Likewise, if there are 64 samples outside of the baseline window, the window expands by one ACD lsb. In order to simplify control, the two counters only reset when 512 or 64 respectively is reached. The window expands faster than it contracts because there are more baseline values than pulse values. In this data set, the ratio is about 4:1. If the ratio were 1:1 then 50% of the samples would be outside of the window (instead of 20%). This means that for this data set, 30% of the samples would be misclassified as part of a pulse. This ratio may have to change for different event rates and will likely be a user input, but will be uniform for the whole scanner.

This scheme accomplishes many things. First, if the noise level reduces, the window can contract closer to the baseline and reject more of each pulse. Likewise, if the noise increases in amplitude, the window can expand. The other thing that this scheme will handle is a shift in the baseline as shown in Fig. 4. Notice that if a static baseline window were used, a quick shift in the baseline that is large enough to move it out of the baseline window would be a problem. The problem would be that none of the baseline values would be inside the baseline window, so they would erroneously be determined to be a part of an event pulse. With the scheme discussed above, the baseline window would expand until the baseline is again inside the baseline window and then contract around the new baseline.

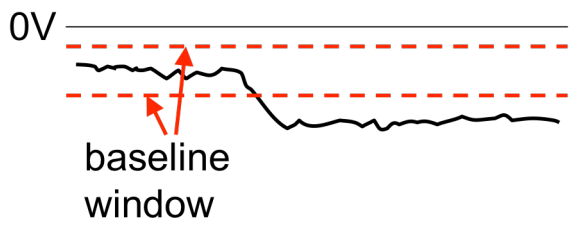


Fig. 4. Illustration of a baseline shift greater than the baseline window.

To calculate the baseline window, the window is initialized to +1 and -1 ADC lsb. The system runs until enough samples have been processed for the baseline window to move half of the ADC range. This assures that the baseline window will settle before the rest of the algorithm starts.

With the baseline window settled, baseline values can be calculated. In this algorithm, a new baseline value is calculated each cycle (ADC cycle). To determine the baseline value while adhering to the requirements previously stated, a running average is utilized. A running average of the last 64 samples allows the baseline to move, but smoothes out the higher frequency noise. When a sample is out of the baseline window, the current baseline value is inserted into the running average. This simplifies the control of the FPGA implementation. Notice, that at this point, pulses cannot be detected because the voltage threshold has not been determined. All is known is whether a sample is inside our outside of the baseline window. The next steps will calculate the parameters necessary to detect pulses.

B. *General Statistics*

In order for this process to be fully automated and suitable to any detectors, all of the pulse parameters necessary for this algorithm and pulse timing need to be automatically calculated. The parameters necessary for these algorithms are a voltage threshold to differentiate pulses from noise, the average pulse length, and the average pulse energy.

It turns out that a good threshold is the baseline window (the half of the window in the direction of photodector pulses). So after the baseline window has settled, 1024 successive baseline window levels are averaged to calculate the voltage threshold for detecting pulses.

Once the threshold is calculated, the average pulse length can be determined. The pulse length is designated as the number of samples above the threshold. When eight consecutive ADC samples are above the threshold, the number of samples (including the initial eight) are counted until the voltage returns back below the threshold. This is again done for 1024 pulses and the average pulse length is calculated. One possible pitfall to this method is the baseline shift as shown in Fig. 4. If this occurs, two things will happen. The pulse length counter will reach the max pulse length count, and this will repeat until the baseline window has expanded to recapture the baseline. This will result in a large portion of the pulse length counts in the average being incorrect. To guard against this, a new pulse is only processed every 65536 cycles. This allows sufficient time for

the baseline window to adjust between pulses that are counted for length.

Determining the average pulse energy is performed in a similar manner. When a pulse is detected, all of the samples of the pulse are summed. The average pulse length determined in the previous step is the number of samples summed for each pulse. Again, this is done for 1024 pulses before the average energy is calculated. As with the pulse length, only one pulse is processed per 65536 cycles so that an anomaly won't be a significant portion of the calculation.

C. Reference Pulse Formation

Once the statistics are calculated, pulses can be detected and processed to form the higher resolution reference pulse. The first step in this process is to detect pulses from the free-running ADC. Just like in the previous steps, this is done by detecting when eight consecutive samples are above the threshold. In addition to using the threshold, a narrow energy quantification around the average pulse energy is also used. This is done by keeping a running sum of the last "average pulse length" samples. The ADC samples are delayed by the same amount so when samples are detected over the threshold, the energy can be immediately checked. The narrow energy window guards against piled-up pulses from being a part of the reference pulse.

Once a pulse is detected, the amplitude must be normalized to a common amplitude. This is accomplished by normalizing the sum of the samples of the pulse. In other words, since the pulses have the same shape, if the area under the pulse is normalized, then the amplitudes will be normalized. This is calculated by using a lookup table to eliminate the division. The summation of the pulse is the input to the lookup table and the output is the ratio that the pulse is multiplied by in order to normalize it. One complication with this lookup table is that in theory, it is required to cover all possible pulse summation inputs. This would make this table prohibitively large. Recall though, that a narrow energy window is used so only a small portion of the table will be required, but which portion isn't known until the average pulse energy is calculated. There are two methods to produce the portion of the lookup table required. The first is to store the whole table on off-chip memory and only read in the required portion. The method we settled on is to use the NIOSII soft microprocessor to calculate the region on the fly.

After the pulse amplitudes have been normalized, the next step is to line them up in time. In order to use the ADC samples to build a higher resolution reference pulse, the relative time of the sampling needs to be determined. Notice that the free running ADC samples asynchronously to the start time of a pulse, so pulses will range from starting just after the ADC sample to starting just before the ADC sample. As illustrated in Fig. 5, this will result in the first sample of the pulse having a range of voltages.

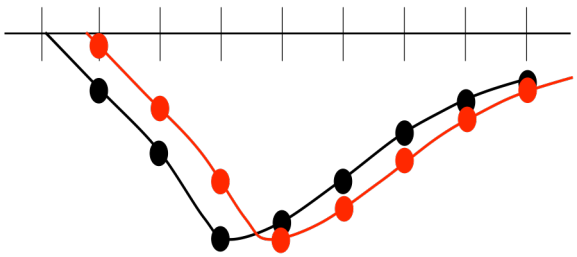


Fig. 5. Illustration of the range of possible sampling of photodetector pulses.

To build the higher resolution reference pulse, each ADC sample interval is divided into 256 sub-intervals. So the sampling interval of 15.4ns (65MHz ADC) is broken up into 256 60ps steps. An array with a length of 256 times the length of pulse in ADC samples is created to sort the sampled pulses into. Locations 0, 256, 512, etc. in the array will be composed of samples from pulses that start just before an ADC sample while locations 255, 511, 767, etc. will be composed of pulses that started just after an ADC sample. To determine which of the 256 sub-intervals to place the samples of a pulse in, the start time of the pulse needs to be determined. Unfortunately, the timing technique we normally use requires a reference pulse. Since this algorithm is in the process of building this reference pulse, an alternative method must be used. For this step, we utilized a linear interpolation. For each pulse, the two samples on either side of the threshold are used to calculate a line and the time of threshold crossing is interpreted using that line. In this case, we designate the crossing of the threshold to be the start time of the pulse.

After 8192 pulses are placed in the array there will be on average 32 samples in each 60ps bin. It is imperative that this is the case, and that there is a linear distribution across the ADC sampling interval. If the samples are not evenly distributed across the ADC interval the shape of the reference pulse will be incorrect. For example, if there are few or no samples in the upper intervals (e.g. 220-255) then the leading edge of the pulse will be too steep because the samples that are higher voltage and were suppose to be in those upper intervals were erroneously placed in the middle buckets. Likewise, if samples that are suppose to be placed in the upper intervals are placed in the lower intervals of the next ADC interval the slope of the leading edge of the pulse will be too shallow. Fig. 6 shows that we were able to get a linear distribution of the samples across the interval. Fig. 6 is a histogram of the first ADC interval with the x-axis being the 60ps bins and the y-axis indicating the number of samples in each 60ps bin.

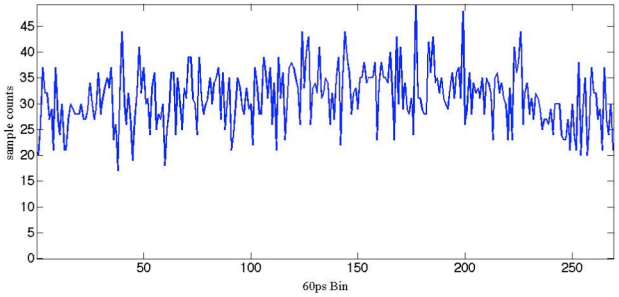
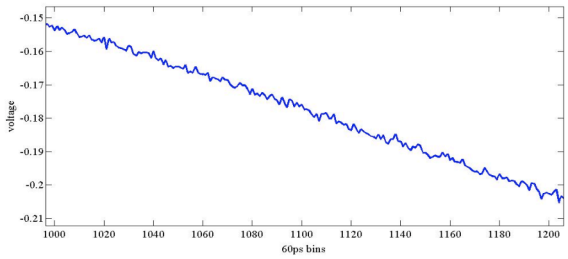


Fig. 6. Histogram of the distribution of pulse samples over one ADC interval (0-255) for the higher resolution pulse.

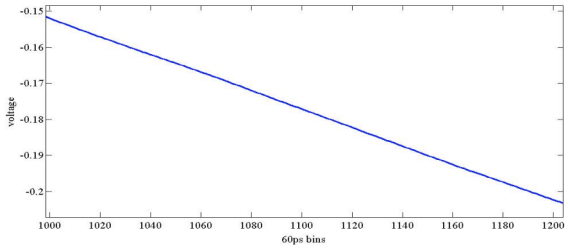
To calculate the reference pulse from the array with multiple samples per 60ps bin is a two-step process. The first step is to find the average value for each bin. To save memory space, a two dimensional array is used to store all of the 8192 sorted pulses. When a sample is to be placed in a particular 60ps bin, the bin read from memory, the sample is added to the read value, and then the sum is written back. A second memory is kept that tracks the number of samples in each bin. When all 8192 samples have been collected, the average of each 60ps bin is calculated by dividing the summation in each bin by the number of samples indicated in the second memory. After this step, the pulse is still very rough as shown in Fig. 7(a). To smooth the pulse, a moving average filter is used. Each bin is recalculated using the smoothing equation

$$v_i' = \frac{1}{49}\sum_{i-24}^{i+24} v_i . \qquad (1)$$

The averaging is done over 49 samples so there is the same number of samples on either side of the current point. In the FPGA, the division by 49 is converted into a multiplication of the reciprocal.



(a)



(b)

Fig. 7. Matlab plot of 12ns of the leading edge of the derived reference pulse (a) before smoothing and (b) after smoothing.

One side effect of using a linear interpolation to determine the start time of the pulse is that the initial portion of the reference pulse (first ADC interval) becomes a line. Also, a linear interpolation is not a very accurate timing model. To alleviate this, the process is iterated three more times. The only difference with these iterations is that instead of using a linear interpolation to determine the start time, the reference pulse formed from the previous iteration is used in the timing technique discussed in section II. Note that the baseline window and general statistics do not need to be recalculated. Each iteration collects a new set of 8192 pulses and forms a new reference pulse. One subtle issue with using the reference pulse to interpolate the start time of the pulse is that if only one point is used as in our normal timing algorithm, the first section of the resulting reference pulse will be an exact match of the previous reference pulse. To eliminate this issue, the first four pulses on the leading edge of the pulse are timed using the voltage to time lookup table. The start time of the pulse is the average start time returned for each of the four samples.

IV. Results

To determine how well this method worked, two experiments were conducted. The data sets for these experiments consisted of pulses sampled from four different Zecotech MAPDN photodetectors using LFS-3. The pulses from a 511 KeV (^{22}Na) source were collected with a 25GSPS oscilloscope and imported into Matlab. Two reference pulses were computed for each data set (each separate detector). One reference pulse was derived by using the data from the oscilloscope that was sampled at 25GHz. The pulses where normalized for amplitude and then an average pulse was calculated. The other reference pulse was calculated using the algorithm discussed above with pulses that were down sampled from 25GHz to 65MHz. To test the ADC sampling rate affect on this algorithm, a reference pulse was also calculated using pulses down sampled to 125MHz.

The first experiment is a subjective comparison of the two reference pulses – the hand-tuned one from 25GHz oscilloscope data and the reference pulse derived from our algorithm. Fig. 8 indicates that pulses can be derived from low-speed ADCs data using our algorithm. The minor inconsistencies at the very start and end of the pulses are of very little consequence to the timing algorithm. The differences at the start of the pulse are below the event threshold so no samples will be processed in that range. The end of the pulse is only utilized in the area normalization step and again most of the differences are below the threshold and so they will not be included in the pulse energy summation.

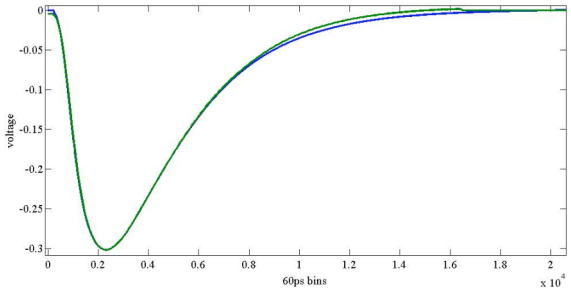


Fig. 8. Matlab plot of two reference pulses. The green pulse is derived from our algorithm while the blue pulse is derived from 25GHz oscilloscope data.

The second experiment was to compare the timing resolution for the data sets using the reference pulses derived in the two different manners. The four data sets were paired up in all possible combinations and coincidental pairs where simulated by aligning pulses in each data stream. Each stream was timed using the reference pulse derived from itself. Fig. 9 shows the results of this experiment. Two ADC sampling rates were investigated – 65MHz and 125MHz. The blue bars indicate the timing resolution for our timing algorithm when the reference pulse is derived using the algorithm discussed above. The grey bars indicate the timing resolution when the reference pulse is the hand-tuned pulse from 25GHz data. For the 65MHz group, the pulses being timed and the pulses used to form the reference pulse with our algorithm are sub-sampled at 65MHz. The hand-tuned reference pulse is still formed with 25GHz data. Similarly, the 125MHz group used pulses sub-sampled at 125MHz for the pulses to be timed and the input to the reference pulse discovery algorithm. With a 65MHz ADC sampling rate, there is no difference in the timing resolution. With a 125MHz there is a 5% degradation in timing resolution using the automated reference pulse.

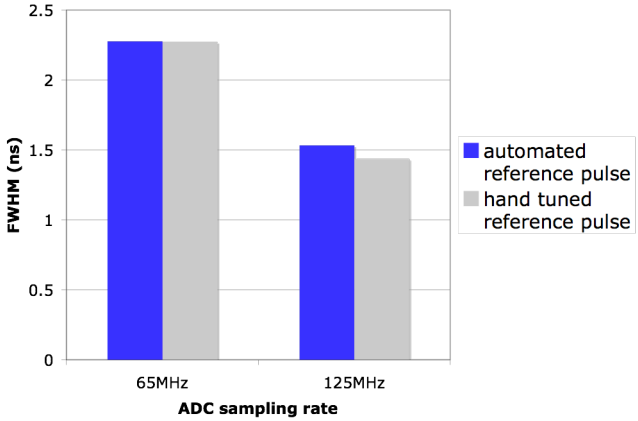


Fig. 9. Result of coincidental timing for the hand-tuned reference pulse derived from 25GHz oscilloscope data (grey) and a reference pulse from out automated algorithm (blue).

Finally, to determine how the accuracy of the reference pulse affects the timing resolution, an experiment was conducted using the "wrong" reference pulse. A simulation was performed where the data sets were timed using reference pulses formed from other data sets. The largest degradation in timing resolution was 35%. This indicates that each detector will need to form its own reference pulse.

V. FPGA Implementation

In order to determine the feasibility of an FPGA implementation, this algorithm was implemented in Verilog and compiled with Altera's Quartus II design software. The targeted FPGA was the Stratix III S200 (same FPGA that will be used in our new data acquisition electronics). This algorithm uses about 10% of the logic and about 90% of the dedicated memory. The high memory usage is due to the storage of the intermediate reference pulses. Notice that two references pulses have to be stored – one for the reference pulse from the previous iteration and the one that is being formed during the current iteration. Using most of the memory on the chip is not an issue because this is only a tuning circuit and does not need to be present on the FPGA during an acquisition scan. While this work utilizes a larger FPGA, this algorithm would also work on smaller FPGAs with the use of external memory as the logic requirements are small and the throughput is not important as will be discussed next.

The clock for the design can run up to 210MHz. Even though the clock rate is sufficient to process the ADC samples in real time, it should be noted that the overall design couldn't be pipelined sufficiently to process pulses in real time. For example, when the sampled pulses are being written to the 60ps array, it takes one cycle to read the current sum in the array, another cycle to add the current sample to the sum and finally a third cycle to write the new value back to the array. This low throughput is acceptable for two reasons. First, it is possible to drop pulses because this data is only used for tuning and not an image reconstruction. Also, recall that our algorithm intentionally skips pulses to eliminate the possibility of a short anomaly corrupting a large portion of the data set used to build the reference pulse.

VI. Discussion

In this work, we have designed algorithms to automate the acquisition of the necessary pulse parameters for an FPGA-based data processing PET front-end electronics. We have shown that it is possible to acquire the necessary pulse parameter to fully automate this discovery algorithm as well as the timing algorithm. We have also demonstrated that an accurate higher resolution reference pulse that has the same shape of the photodetector pulses can be built out of lower resolution pulses. We have also shown that this algorithm can reasonably be implemented on an FPGA. This work will greatly reduce the need for operator calibration of the system. Additionally, we believe that this method will produce better pulse parameters for our timing algorithm and future pulse-pileup correction algorithms. Particularly since the data that the FPGA processes contains all of the shaping information of the data acquisition chain. Finally, these algorithms will aid in the research of new PET scanner hardware, as any changes in the system can be quickly calibrated with the FPGA.

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. 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*, 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*, 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*, 2007, pp. 3161-3165.

[5] Guerra, P. *et al*., "Digital timing in positron emission tomography," *IEEE Nuclear Science Symp. Conf. Record*, 2006, pp. 1929-1932.

[6] Fontaine, R. *et al*., "Timing Improvement by Low-Pass Filtering and Linear Interpolation for LabPET Scanner," *IEEE Trans. on Nuclear Science,* vol. 55, no. 1, 2008, pp. 34-39.

[7] M. Haselman, D. DeWitt, T. K. Lewellen, R. Miyaoka, S. Hauck, "FPGA-Based Front-End Electronics for Positron Emission Tomography", *ACM/SIGDA Symposium on Field-Programmable Gate Arrays*, pp. 93-102, 2009.