FPGA-Based Pulse Processing for Positron Emission Tomography

Michael Haselman

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2011

Program Authorized to offer Degree:

Electrical Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Michael Haselman

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Co-Chairs of the Supervisory Committee:

____
Scott Hauck

____
Thomas Lewellen

Reading Committee:

____
Scott Hauck

____
Thomas Lewellen

____
Brian Nelson

Date: ______________________________

University of Washington

Abstract

FPGA-Based Pulse Processing for Positron Emission Tomography

Michael Haselman

Co-Chairs of the Supervisory Committee:
Professor Scott Hauck
Electrical Engineering

Professor Thomas Lewellen
Radiology

Data acquisition and pulse processing for positron emission tomography, or PET, has traditionally been an analog domain due to the unique performance requirements of the scanner hardware. Foremost on the performance requirements is the ability to timestamp the photodetector pulses with very high precision. This thesis proposes a new timing algorithm that leverages the increasing ADC sampling rates and FPGA computational power to eliminate the need for most of the analog circuits. This will simplify the design and possibly reduce the power of the data acquisition electronics. This thesis also enhances the pulse processing capabilities by adding an all-digital method of correcting for pulse pile-up. Additionally, we utilize the unique features of the FPGA to tune itself to specific pulse parameter of each photodetector in order to improve the pulse processing precision.

The algorithms developed in this thesis take advantage of the fairly consistent pulse shape of the photodetector pulses. The general idea is to fit a higher-resolution, predefined pulse to the lower-resolution data pulse from the photodetector. This "reference" pulse, which is built to have the same shape as the data pulses, is used to interpolate the start of the data pulse. This reference pulse is also utilized to separate pulses that are involved in pulse pile-up. The results demonstrate that a well-defined FPGA timing circuit can match the performance of an analog circuit while using relatively few FPGA resources. We also demonstrate that the computational power of FPGAs can be uses to make the signal processing more robust in the presence of pulse pile-up.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Graduate school has been a long and sometime difficult journey for me, and I would like to acknowledge some of the people who encouraged and supported me to the end as a small token of my immense appreciation.

First and foremost I need to thank my wife, Amy, for her support as I plodded my way through graduate school. She has shown great patience and sacrificed a lot while I worked towards my goals. I am also grateful for support and encouragement from my parents and in-laws. Their help, especially with the three children we had during graduate school, has been invaluable. Finally, I'm thankful for my kids Kade, Ella, and Asher. They kept me grounded and always put things in perspective

After family, the most important person in the life of a graduate student is his advisor. In this matter, I won the advisor lottery. I have been fortunate enough to work under the advise of Professor Scott Hauck for eight years. He has been patient when I wasn't sure what I wanted to do, supplied the right amount of pressure when the procrastination became too much and always had great insights for my difficulties. These insights were often for questions about life as well as research. I have also been fortunate to have to other advisors from the Radiology Department, Thomas Lewellen and Robert Miyaoka. These two have taught me what the important problems are in PET research.

In addition to academic support, I have relied on the help and encouragement of a number of staff members at the University of Washington. When I decided to go back to school, the electrical engineering undergraduate advisor, Amy Feldman-Bawarshi guided me through the whole process, but more importantly, made me believe I could do it. When it came to collecting data or performing experiments in the radiology laboratory for this thesis, Wendy McDougald was always willing to lend her expertise and time. Pam Eisenheim, our lab program manager, was always there to help with everything else, from getting paid to travel arraignments for conferences.

The money that Pam doled out was graciously provided by Zecotek Photonics, Altera Corporation and the NIH. Without their financial support, graduate school would not have been possible.

DEDICATIONS

I would like to dedicate this work to my family,
who have provided immense support and patients throughout this journey

Chapter 1 : Introduction

The ability to produce images of the inside a living organism without invasive surgery has been one of the major advancements in medicine over the last 100 years. Imaging techniques such as X-ray computer tomography (CT) and magnetic resonance imaging have given doctors and scientists the ability to view high-resolution images of the anatomical structures inside the body. While this has led to advancements in disease diagnosis and treatment, there is a large set of diseases that only manifest as changes in anatomical structure in late stages of the disease, or never at all. This has given rise to another branch of medical imaging that is able to capture certain metabolic activities inside a living body.

The most common forms of functional imaging are positron emission tomography (PET) and single-photon emission computed tomography (SPECT). Both of these techniques use radioactive "tracers" to track metabolic activities. Tracers consist of a metabolically active molecule, such as glucose derivatives, chemically bonded to a radioactive isotope. Once the tracers are injected into the body, they are selectively metabolized by certain parts of the body and therefore achieve higher concentrations in certain tissues. This results in higher radioactivity in these tissues, which can be detected by the scanner and converted into an image. While many of the topics discussed in this paper are relevant to many applications including SPECT and PET, we focus on PET scanners as the motivation and application.

A PET scanner consists of three main systems: the detectors, the data acquisition and pulse processing electronics, and the image reconstruction system. This thesis will concentrate on the data acquisition and pulse-processing portion of the scanner. Traditionally, the data acquisition chain of a PET scanner has been a mix of analog and digital circuits. The digital circuits have generally been used for glue logic, while the analog components do the more complex signal processing steps. Recently, there has been a lot of interest in a fully digital PET scanner [1,2,3,4,5] to take advantage of the improvements in digital technologies. The latest generation analog-to-digital converters (ADCs) and field programmable gate arrays (FPGAs) are well suited to replace, as well as add some functionality, to the signal processing capabilities of the PET scanner.

The capabilities and quality of the pulse processing algorithms have a large influence on the quality of the final PET images. One factor is the ability to precisely timestamp the radioactive events. The current state of the art uses analog circuits to perform the timing [6]. While these analog circuits can achieve sub-nanosecond timing resolution, FPGAs have made advancements in computing power and I/O sophistication that may allow them to achieve similar or better timing results. Since PET scanners can have hundreds of data channels per printed circuit board (PCB), board area and power consumption are a big concern. Many current PET systems already utilize FPGAs for data acquisition [2,3], so it is logical to employ the already used circuit board area to compute the timing pickoff, if suitable precision can be achieved. In this thesis, we present an all-digital timing method that achieves suitable timing precision while maintaining the necessary throughput to process data from the ADC in real time.

An advantage FPGAs have over analog circuits is their ability to adapt to the variations in the different detectors of the scanner, including temporal changes that may occur. Utilizing FPGAs as the main processing platform allows for periodic tuning of the system by reconfiguring the FPGA to perform the appropriate calibration circuit. This is important because the individual parts of the scanner don't behave the same, and they also tend to drift over time. The ability to tune the scanner without extra hardware means that the scanner will behave the same over time and there won't be large differences in data obtained from different portions of the scanner. In chapter 5, we will present how we utilize the FPGA to calculate certain pulse parameters that are necessary for the pulse processing steps. This will lead to improved results as well as improve the scanner usability by automating the calibration process.

Another factor in image quality that is attributable to pulse processing is information that is lost due to pulse pileup. Pulse pileup occurs when two pulses overlap. If no steps are taken to separate the pulses, the data contained in the pulse is either lost or adds noise to the system. There are some analog [7,8] and mixed analog and digital [9] solutions to correct for pulse pileup. An all-digital solution would improve the overall system architecture, as no external analog circuits would be required. Chapter 6 will discuss how the circuit designed for the timing function can be modified to handle pulse pileup.

This dissertation discusses the development of an all-digital pulse processing suite for a PET scanner. The algorithms developed in this work are specifically designed to leverage the strengths of FPGAs in order to improve on the analog equivalents. It is organized as follows:

- **Chapter 2: Positron Emission Tomography** introduces the topics of PET including the tracer chemistry, the scanner hardware, the image reconstruction and the uses of PET.
- **Chapter 3: Field Programmable Gate Arrays** covers the architecture of modern FPGAs and why they are a good solution for the data acquisition electronics of a PET scanner. The strengths and weaknesses of competing technologies are also discussed to further motivate the decision to use FPGAs as the computing device.
- **Chapter 4: Timing** discusses the existing techniques for placing accurate timestamps on photon arrival times and presents a novel all-digital timing algorithm developed specifically for the FPGA.
- **Chapter 5: Pulse Parameter Discovery** describes a tuning circuit that was developed to support the timing algorithm as well as other pulse processing algorithms that will be present in the FPGA.
- **Chapter 6: Pulse Pileup Correction** introduces the issues of pulse pileup and presents some modifications to the timing algorithm that will allow some of these pulses to be processed correctly.

- **Chapter 7: MiCEs Scanner Hardware and Overall Processing Architecture** presents the hardware of the small-animal PET scanner we are developing. It discusses the overall architecture of the circuits in the FPGA and how they will communicate with the scanner hardware.
- **Chapter 8: Conclusions and Future Work** summarizes the contributions of the dissertation and present some possible future work.

Chapter 2 : Positron Emission Tomography

PET is a medical imaging modality that uses radioactive decays to measure certain metabolic activities inside living organisms. It does this through three main components (Figure 2.1). The first step is to generate and administer the radioactive tracer. A tracer is made up of a radioactive isotope and a metabolically active molecule. These tracers are injected into the body of the patient to be scanned. After enough time has lapsed for the tracer to distribute and concentrate in certain tissues, the subject is placed inside the scanner. The radioactive decay events for tracers used in PET studies give off two 511keV antiparallel photons. The scanner hardware is designed to efficiently and accurately capture these photons. The scanner, consists of a ring of sensors attached to electronics. The sensors are made up of scintillator crystals attached to a photodetector. The scintillator converts the 511keV photon into many visible light photons, while the photodetector generates an electrical pulse in response to the burst of visible light. These pulses are processed by the front-end electronics to determine the parameters of each pulse (i.e. energy, timing). Finally, the data is sent to a host computer that performs the tomography to turn the data into a 3-D image.

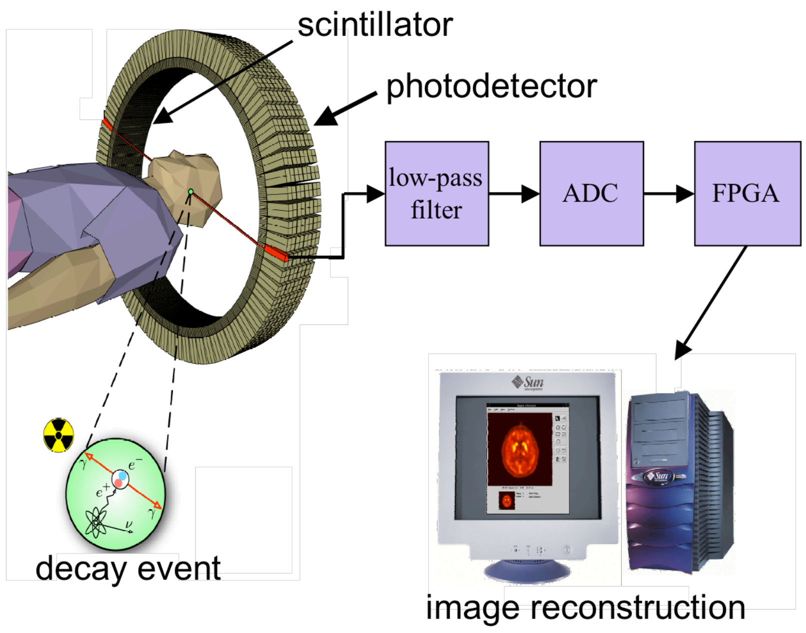


Figure 2.1. Drawing of a PET scanner ring and attached electronics.

2.1: Radiopharmaceutical

The first step in a PET scan is to create the radiopharmaceutical (or tracer). To synthesize the tracer, a short-lived radioactive isotope is attached to a metabolically active molecule. Generally, the isotope replaces a small portion of the molecule. For example, in the glucose analog FDG, a hydroxyl group is replaced by fluorine-18. The metabolically active molecule acts as a transport for the radioactive isotope to target the tissue of interest. The short half-life means that a large portion of the decays will occur during the scan period, which maximizes the information gathered for a given dose of radiation. It also means that the exposure to ionizing radiation is essentially limited to the timeframe of the scan. Unfortunately, it also means that the tracer has to be produced close to the scanner, as it does not store for very long. This makes PET traces expensive and limits PET scanners to locations close to the facilities to produce tracers. Some common tracers and their targeted tissue are shown in Table 2.1.

Table 2.1. Half-life and typical application of selected radiopharmaceuticals.

Tracer	Half Life (min)	Imaging Application
[O-15] water	2	Brain Blood Flow
[N-13] ammonia	10	Myocardial Perfusion
[C-13] acetate	20	Myocardial Flow and Metabolism
[F-18] FDG	110	Brain, Heart, & Tumor

The most commonly used tracer is fluorine-18 flourodeoxyglucose ([F-18]FDG). [F-18]FDG is a derivative of glucose that has a half-life of 110 minutes (plutonium-239 in comparison has a half life of 88 years). [F-18]FDG is similar enough to glucose that cells that utilize glucose still consume it, but the modifications are such that it cannot be completely broken down and excreted from the cell until the isotope (fluorine-18) decays. This means that cells that consume glucose will have high concentration of the tracer. For example, it turns out that cancerous cells consume proportionally more glucose than normal surrounding tissue. Thus, when a patient has a PET scan with [F-18]FDG, cancerous tissue will show up in the final image as spots of greater radioactive decay activity.

2.2: Decay Event

After sufficient time has passed for the tissue of interest to uptake the tracer, the scanner is used to detect the radioactive decay event (Figure 2.2). The isotopes used in PET are positron emitters. When the isotope undergoes positron emission decay (also called positive beta decay), a proton changes to a neutron and emits a proton and nuetrino. This proton travels a very short distance in tissue (<2mm) as it loses kinetic energy. Once it loses enough energy, it interacts with an electron. The interaction results in the annihilation of the photon and electron. Because of the conservation of momentum and energy, the result is the generation of two 511keV (the resting energy of an electron) photons that travel away from the annihilation at 180° (± .23°) from one another.

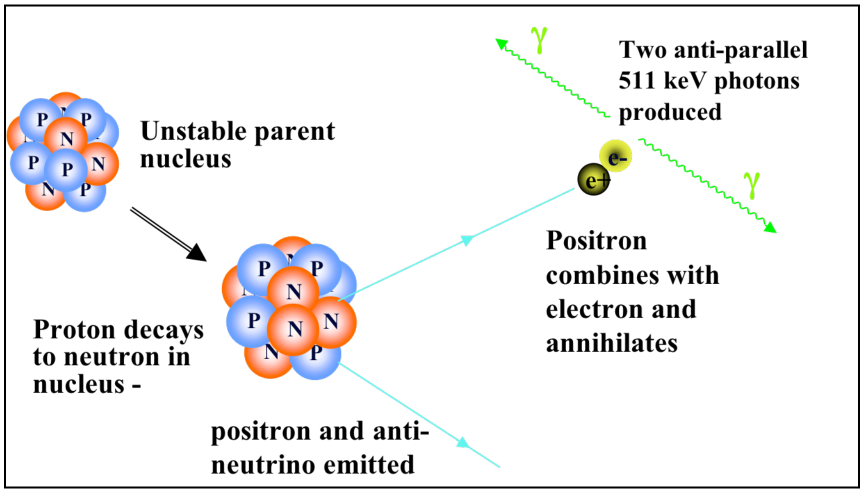

Unstable parent nucleus
Proton decays to neutron in nucleus -
positron and anti-neutrino emitted
Two anti-parallel 511 keV photons produced
γ
γ
e-
e+
Positron combines with electron and annihilates
P
N

Figure 2.2. Physics of a positron decay and electron annihilation [11].

2.3: Photon Scintillation

A 511keV photon has a substantial amount of energy, and will pass through many materials, including body tissue. While this is beneficial for observing the photon outside the body, it makes it difficult to stop the photon for detection. This is the task of the scintillator. A scintillator is a crystal that absorbs high-energy photons and then emits photons in the visible light spectrum in response. Scintillators can be made up of many different materials, including plastics, organic and inorganic crystals, and organic liquids. Each scintillator has a different density, wavelength of maximum emission, and timing characteristics. The density or linear attenuation determines how well the material stops the photons, but also affects how much light will be emitted from the crystal surface. As more light is emitted, the signal to noise ratio improves. Thus there is generally a tradeoff between the probability a photon passing through the crystal will be absorbed and how much visible light is emitted. Ideally, the peak wavelength of the emitted light is matched with the absorption spectrum of the device that converts the light into an electrical pulse to increase the light collection efficiency. Finally, the timing characteristics indicate how long it takes for the visible light to reach its maximum output (rise time) and how long it takes for the light pulse to decay (decay time). A shorter rise time generally produces better timing resolution. The length of the pulse is important because the longer the pulse, the lower the number of events per second a scintillator can handle. A lower count rate requires a longer scan to get the same number of counts. One common scintillator material is $Lu_2SIo_5(Ce)$, or LSO, which is an inorganic crystal; it has a reported rise constant of 30ps and a decay constant of 40ns [10]. These documented times can vary in practice due to the geometry of the crystal and the electronics that are attached to it.

2.4: Photodetectors

Attached to the scintillator are electronic devices that convert the visible light photons into electronic pulses. There are two fundamental technologies used for photomultipliers. Currently, the most commonly used device is a photomultiplier tube (PMT), which is a vacuum tube with a photocathode, several dynodes, and an anode that have high gains to allow very low levels of light to be detected. There are many disadvantages of PMTs however. The first is they require a high bias voltage, and thus use a lot of power. They are also bulky and cannot operate in high magnetic fields, which precludes using them as an insert into a magnetic resonance imaging (MRI) scanner (a current focus of medical imaging research).

Emerging photodetector technologies are based on solid-state detectors. While they have a variety of names such as avalanche photodiodes (APDs) or silicon photomultipliers (SiPMs), they rely on essentially the same technology. They are based on a reversed-biased diode that breaks down and produces a current when struck by a photon in the visible light spectrum. The most promising detector, the SiPM, is an array of semiconducting photodiodes that operate in Geiger mode. A common arrangement is an 8x8 array of about 3 mm x 3 mm cells. Each cell has a density of about 10^3 diodes per mm^2. All of the diodes in a cell are connected to a common silicon substrate so the output of the array is a sum of all of the diodes that "fire" (Figure 2.3). The output can then range from one diode firing to all of them firing. This gives these devices a linear output, even though they are made up of discrete diodes.

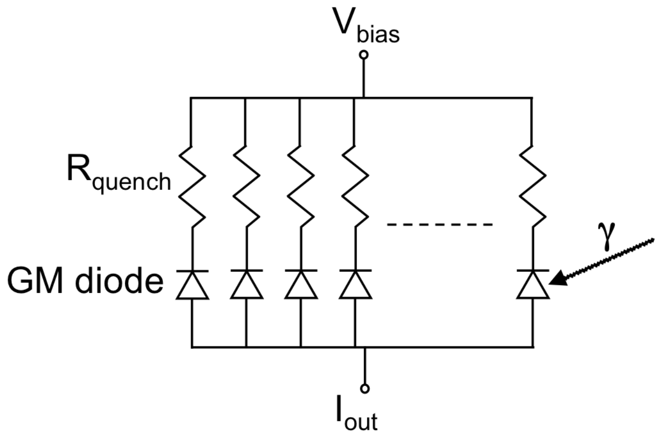


Figure 2.3. Circuit diagram of SiPM array.

The main advantages of SiPMs are that they are very compact, they are lower power than PMTs, they function in a magnetic field, and they are fabricated with a CMOS process. The ability to fabricate these detectors with a CMOS process has multiple advantages, including reduced costs and increased reliability. Another advantage that is the subject of current research is the integration of the data acquisition and pulse processing electronics on the same substrate [12,13]. This eliminates issues with sending the signal to a separate chip, such as parasitic capacitance of interconnects and the addition of noise. Additionally, instead of treating the output of a cell as an analog signal, each diode in the cell can be treated as a digital device (either it fired or it didn't).

2.5: Front-End Electronics

The pulses from the photodetector (Figure 2.4), along with their location and start time of interaction with the scanner, contain most of the information needed to create a PET image. However, the pulses need to be processed to extract that information. This is done with the front-end electronics. The front-end electronics consist of a filter, ADC, and FPGA in our system (see Figure 2.1), but traditionally the electronics have been a mix of analog parts, microprocessors and FPGAs. The information that needs to be calculated for each pulse is the pulse energy, start time and event location in the crystal array. Additionally, this information may have to be acquired in the presence of pulse pileup. This complexity, which will be addressed in Chapter 6, is when a second pulse starts before the first pulse has returned to its baseline.

To process the output of the photodetectors, the pulse first needs to be filtered to remove some of the noise (Figure 2.4). This is done with a low pass filter. Then, before the analog pulses can be processed by the FPGA, it is digitized with an analog to digital converter (ADC).

Now that the data is digitized, the needed parameters can be extracted in the FPGA. Ignoring the complication of pulse pileup (chapter 6), the easiest data to extract is the total energy. The straightforward way is to sum up the samples of the pulse values and subtract out the baseline (the output value of the photodetector without any input stimulus). This method, while simple, can reduce the energy resolution of the system because it relies on a small number of noisy data points (with a 65MHz ADC, pulses in our system are roughly 10 samples long). A more precise method uses pulse fitting to extract the energy value [1].

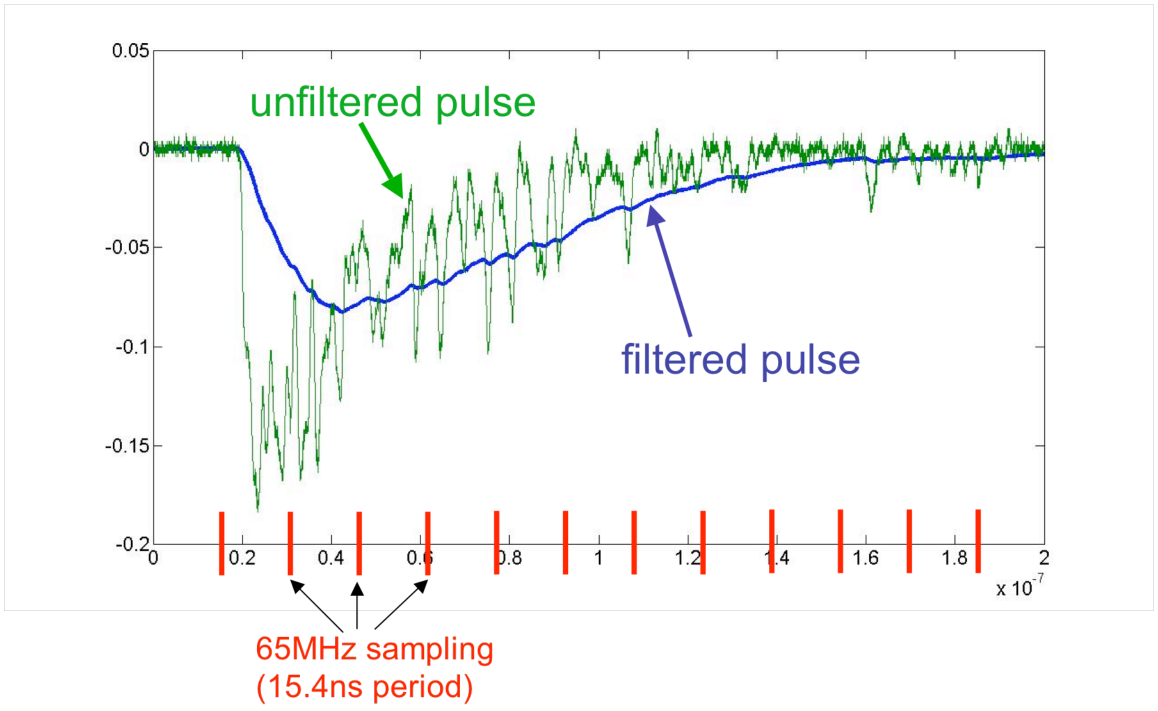


Figure 2.4. MATLAB plot of a pulse from a LSO scintillator crystal attached to a PMT. A simulated low-pass filter with a 33.3MHz cutoff is also shown. The marks on the x-axis indicate the interval at which 65MHz ADC would sample.

Determining the start time of the pulse (topic of Chapter 4) is important for detecting coincidental pairs. Coincidental pairs refer to the two photons that arise from a single radioactive decay event. Many of the photons from a radioactive event don't reach the scanner because they are either absorbed or deflected by body tissue, they don't hit the scanner, or the scintillator crystal does not stop them. So the scanner must have a method for determining if both photons interacted with the scanner. A PET scanner's main advantage over single photon emission computed tomography (SPECT) is that it works by detecting two photons from an event and essentially draws a line that represents the path of the photons. SPECT tracers only emit one photon and use mechanical techniques, such as lead collimators, to determine the line of travel. This means that only the photons that are traveling perpendicular to the SPECT camera are utilized, while the rest are absorbed by the lead. PET on the other hand can utilize events that hit the scanner at any angle, so long as the other photon from the same event also interacts with the scanner. However, if only one of the two emitted photons hits the PET scanners there is no way to determine where the event occurred. To determine that two photons are from the same decay event, they have to interact with the scanner within a certain time of each other and they need to be within the field of view (FOV), as shown in Figure 2.5. If interactions are detected in detectors A and B within a certain time period, then the interactions are saved as a coincidental pair and are considered to have arisen from the same decay event. If interactions are detected in detectors B and C, then the interaction is ignored because it is out of the field of view (FOV), which is represented by the dotted lines. Likewise, if interactions are detected in detectors A and B but with a large time separation, they are ignored; Considering that the photons travel at essentially the speed of light, two photons from a single decay event will hit the opposite sides of the scanner almost instantaneously. The process of comparing the arrival times of two photons is done by placing a very precise time stamp on each event. The two timestamps are compared, and if they are within a certain time of each other they are considered a coincidental pair (assuming the FOV is satisfied). The better the precision of the time stamp, the smaller the coincidence window can be, which lowers the probability that two separate random photons will be paired together. The FOV for one detector (a detector is defined as a scintillator/PMT pair) consists of a set of detectors on the opposite side of the scanner, so any photons that originate from the object being scanned will fall in the FOV. If the FOV is set too large, events that have one photon scattered could be erroneously determined to be coincidental, (detectors B and C in Figure 2.5). If the FOV is too small, events that originate on the periphery of the patient may be rejected as non-coincidental.

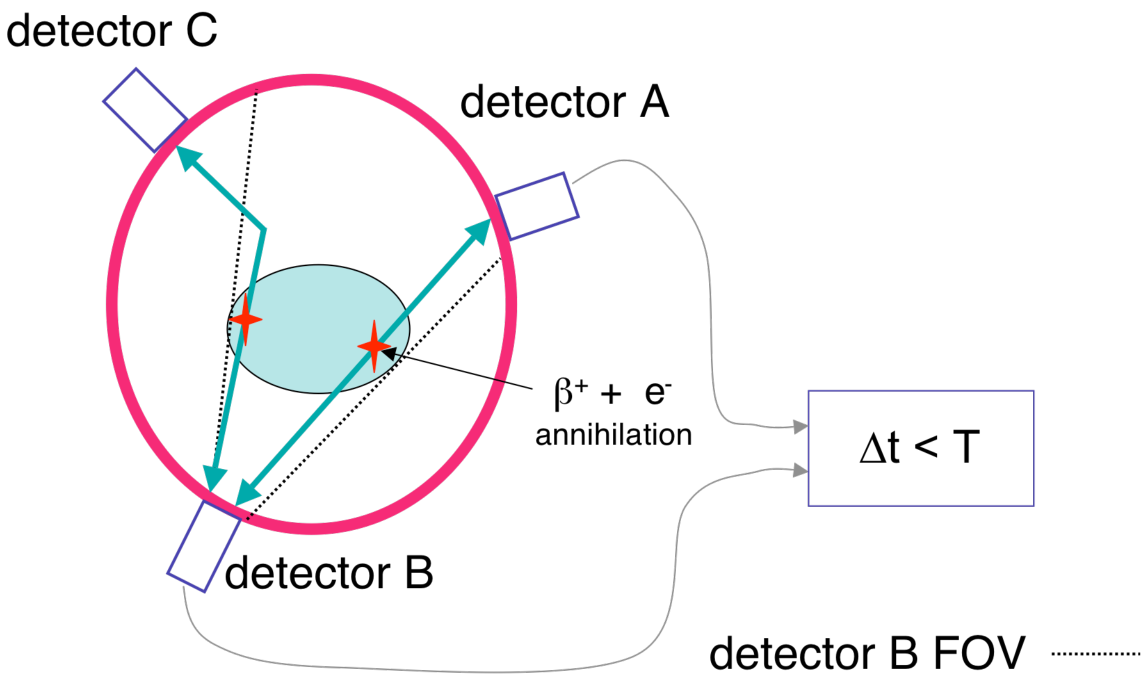


Figure 2.5. Coincidence detection in the PET scanner. [14]

2.6: Image Reconstruction

When all coincidental events have been sent to the host computer, image reconstruction can begin. The details of image reconstruction are beyond the scope of this thesis, but essentially all of the events are separated into coplaner lines of response (interpreted path of the photon pair). These lines can then be used to create a number of 1-D image, which can then be back projected to create a 2-D image, as shown in Figure 2.6, using computer tomography. In addition to the projections, additional steps are taken to increase the image quality.

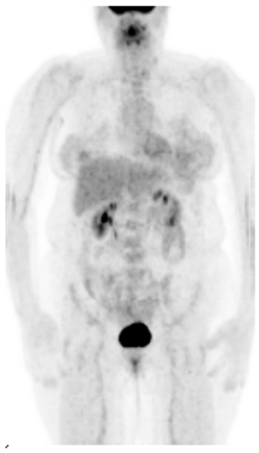


Figure 2.6. Whole body PET scan image using [F-18]FDG. Notice all of the body has some radioactivity because glucose is utilized in many cells. Notable "hot" spots are in the abdomen, the bladder near the bottom of the image (excess glucose is excreted in the urine), and the brain.

2.7: Uses of PET

While PET, magnetic resonance imaging (MRI), and computed tomography (CT) are all common medical imaging techniques, the information obtained from each of them is quite different. MRI and CT are generally utilized to acquire anatomical or structural information. That is, they produce a high-resolution picture of the anatomical structure of the body. This is great for problems such as broken bones, torn ligaments or anything else that presents as abnormal structure. However, it doesn't give any indication of metabolic activity. This is primarily the domain of PET. The use of metabolically active tracers means that the images produced by PET provide functional or biochemical information.

Oncology (study of cancer) is currently the largest use of PET. Certain cancerous tissues metabolize more glucose than normal tissue. [F-18]FDG is close enough to glucose that cancerous cells readily absorb it, and therefore they have high radioactive activity relative to background tissue during a scan. This enables a PET scan to detect some cancers before they are large enough to be seen on an MRI scan. They are also very useful for treatment progression, as the quantity of tracer uptake can be tracked over the progression of the therapy [15]. In other words, the number of events detected from cancerous tissue is directly related with the tracer uptake of the tissue. So, if a scan indicates lower activity in the same cancerous tissue after therapy, it indicates the therapy is working. There are many other uses for PET including neurology (study of the nervous system) and cardiology (study of the heart). An interesting application in neurology is the early diagnosis of Parkinson's disease. Tracers have been developed that concentrate in the cells in the brain that produce dopamine [10], a neurotransmitter. In patients with Parkinson's disease, neurons that produce dopamine reduce in number. So, a scan of a patient with Parkinson's would have less activity than a healthy patient. This can lead to early diagnosis, since many of the other early signs of Parkinson's are similar to other diseases. It is also useful to see if therapies are progressing. If a therapy results in more cells that produce dopamine, then there will subsequently be more activity than previous scans for the same patient.

Chapter 3 : Field Programmable Gate Arrays

FPGAs are reprogrammable semiconductors that can be configured to perform almost any arbitrary digital computation. This programmability, along with low engineering costs, make FPGAs a superior solution for many embedded computing applications. While there are competing technologies, FPGAs have found a large enough niche to create a multi-billion dollar industry. This chapter will cover the architecture and general method of computation for modern FPGAs. It will also discuss the strengths and weaknesses of FPGAs as well as competing technologies. Finally, the requirements of the PET data acquisition hardware will be examined to determine why the strengths of FPGAs make them a great solution for this domain.

3.1: FPGA Architecture

FPGAs perform computations through an array of small computation elements that are connected by a large reconfigurable network (Figure 3.1). The computational elements are generally small memories that have four to six inputs and one output. These small memories, called Look-Up Tables (LUTs), can perform any n-input (typically 4 or 6), one output Boolean function by configuring the contents of the memory. For example, by configuring a '1' at the largest address location (address '1111' for a four input LUT) and a '0' at all other addresses, an n-input AND gate can be created. The LUTs are combined together with other logic, such as internal routing, fast carry chains, multiplexers and flip-flops, to create a logic block. The logic blocks are connected with a configurable interconnect network to build more complex computations. The computation size and complexity is only limited by the size of the logic array and the restrictions of the interconnect. Also included in most modern commercial FPGAs are dedicated blocks such as phase-lock loops, fast I/O transceivers, random access memories (RAMs), and multipliers. While the multipliers and the memories can be implement in the logic blocks, using dedicated blocks results in a substantial area and power savings, as well as increased execution speed. The possible downside of these dedicated blocks is that if they are not utilized, they take up silicon and cannot be programmed to perform other logic like a LUT. Since memories and multipliers are very common in traditional FPGA problem domains, the inclusion of these dedicated blocks makes perfect sense.

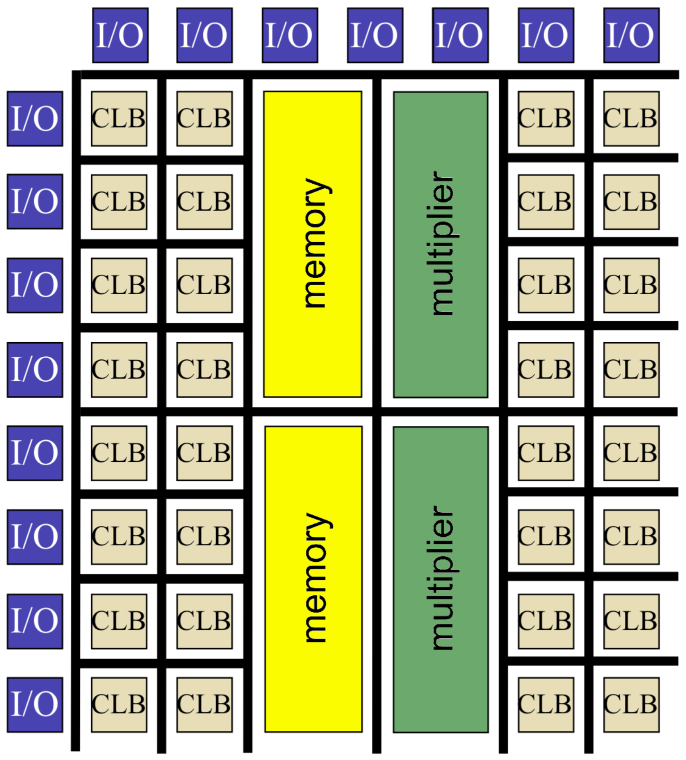


Figure 3.1. Architecture overview of a typical commercial FPGA.

3.2: FPGAs, Microprocessors and Application Specific Integrated Circuits

The main competing technologies in the embedded domain are microprocessors and application specific integrated circuit (ASICs). For certain computations, FPGAs can outperform microprocessors because they can compute in parallel and can be configured to perform the algorithm in a custom circuit. Microprocessors on the other hand must compile the algorithm into the instruction set of the processor. This series of instructions are then essentially processed one at a time. For example, if an image processing algorithm requires a constant to be added to every pixel of an image, a microprocessor will have to fetch the same instruction and the pixel value for every pixel and compute the sums one at a time. An FPGA on the other hand can implement hundreds of separate adders, and because there are no dependencies between the data hundreds of sums can be done in one cycle. The number of adders in this example would only be limited by the size of the FPGA and the bandwidth to memory. Even though an FPGA clock cycle is typically about ten times longer than a microprocessor cycle (2GHz vs. 200MHz), it is easy to see circumstances where an FPGA would have much higher performance than a microprocessor. Another domain where FPGAs hold an advantage over microprocessors is when the operators are less than the standard word size (32 or 64 bits). For example, if an algorithm only needs four bits of precision, a microprocessor still has to use the whole 32-bit data path, where the FPGA only has to dedicate a four bit wide data path. An even better example is for bit-wise operations such as the bit permutations used in encryption. To accomplish the same task in a microprocessor would require many cycles. On the other hand, if an algorithm is a long series of interdependent tasks, the microprocessor is going to outperform the FPGA because little parallelism can be extracted. For example, if the computation were a series of instructions where each instruction depended on the result from its previous instruction, there would be no way to create a parallel implementation and the microprocessor's high clock rate will achieve a faster computation. The other advantage that a microprocessor has over the FPGA is code development. The abstraction offered in software development makes writing and debugging the code for processors much easier than developing HDL for FPGAs.

Another alternative computing platform is the ASIC. ASICs are fully custom circuits that are hard-wired to perform a computation, and generally provide the best solution in terms of speed and power. They often represent the leading edge of technology, but they require a large engineering effort to design and manufacture, and therefore only make sense when an enough ASICs will be manufactured and sold to overcome the initial engineering costs. Any computation that can be configured into an FPGA can be done with greater speed and lower power in an ASIC. In fact, many early uses of FPGAs were for circuit verification before it was committed to an ASIC. Lately though, the costs of a new ASIC design have been escalating dramatically, and FPGAs have become a much more attractive solution. Another detriment to ASICs is their lack of flexibility after fabrication; FPGAs can be quickly reconfigured to upgrade code, develop and test new algorithms, or to insert a whole new circuit. This flexibility is an FPGA's biggest advantage, but it also reduces the maximum clock speed and increases the power consumption relative to ASICs.

3.3: Why FPGAs in PET Front-End Electronics?

Many of the strengths of FPGAs discussed above can be utilized in the data acquisition electronics of a PET scanner. Traditionally, data acquisition circuits have been composed of predominately analog circuits. Analog implementations generally work very well, but they have many downsides. They consume a lot of power and can consume a lot of board space. The circuits are also difficult to engineer, which increases development time and costs. Finally, once the circuits are implemented, they are very difficult to modify. This inflexibility is a detriment in a research setting where many different configurations and circuits are prototyped and tested.

The alternative to processing the pulses with analog circuits is to sample the pulses close to the photodetector with an ADC and processes them digitally. As discussed above, the options for digital pulse processing in a PET scanner are essentially microprocessors, ASICs and FPGAs. While a microprocessor may be the easiest platform to program, its limitations make them a bad fit for this application. The limitations that are relevant here are the serial nature of a processor, the rigid I/O structure, and the memory architecture. As discussed above, the microprocessor essentially performs operations in a serial sequence of steps. As will be discussed in the next couple of chapters, there are a number of opportunities to parallelize operations in this application that a microprocessor could not leverage. In the discussion of the overall MiCEs scanner architecture in chapter 7, it will become apparent that the data acquisition electronics require a flexible and unique I/O structure to support the required peripherals (ADCs, memories, system buses, etc.). Finally, the shared memory structure of a processor would hamper the performance of many of the pulse processing steps. A shared memory means that all applications share the same memory, and thus have to take turns accessing the memory. Many of the pulse processing steps that we have developed require separate memories to hold large look-up tables. They also require deterministic access times to assure real-time processing of the data. This could not be assured in a shared memory system where a memory controller would have to arbitrate between the applications to determine who gets access. ASICs on the other hand can implement a circuit in any desired architecture including any circuit the FPGA can implement (ignoring reconfiguration). The main drawback is the engineering costs and inflexibility after fabrication.

FPGAs, in contrast, excel in most of these areas. In the PET pulse processing, there is data-level parallelism (same operation of different data) as well as task-level parallelism (different operations on same data) that can be exploited for increased pulse throughput. The data parallelism comes from the need to operate on many different channels from the photodetector. The same operation has to be done on all of the channels. The FPGA allows us to add enough circuits to process all channels simultaneously. The task parallelism comes from the need to calculate many different values from the same pulse. For example, the start time, energy, and interaction position is needed for each event. To increase the throughput of the system, the pulses can be duplicated and processed for each of these data points simultaneously. Another powerful feature of the FPGA that we can leverage is the user defined I/O. The ability to define the I/O standard and where it is connected in the FPGA is an enormous advantage. It allows us to develop a printed circuit board with exactly the necessary mix of peripherals needed to support the pulse processing applications. These peripherals can be connected directly to the FPGA I/O without using a bus, which provides the deterministic communication delays required. An FPGA's ability to be reconfigured can also be an advantage over ASICs. As will be discussed in chapter 5, we use this feature to implement a periodic calibration algorithm. Also, the reconfigurability can be used to update the circuit to accommodate system changes or upgrades. As discussed in Chapter 7, the hardware was specifically designed to handle multiple detector configurations. The ability to change the functionality in the FPGA makes it possible to use the same boards on different detector designs.

While FPGAs are a powerful computational platform, there are some difficult aspects of using them. One of the difficulties in using FPGAs is the code development. While there are various ways to program a FPGA, using a hardware description language (HDL) like Verilog or VHDL is currently the best way to get the highest performance. This is because HDL allows the designer to control most of the details of an FPGA design. However, this fine-grain control comes at the expense of more complicated development. The abstractions that makes software development for processors easy is generally not available in HDLs. For example, memory accesses have to be explicitly managed in FPGAs, while the system manages them in processors. Although this can make a FPGA engineer's job difficult, it is necessary for the deterministic computing required to implement the algorithms that will be discussed in the following chapters.

Chapter 4 : Timing

One of the PET scanner functions that have traditionally been computed with discrete analog parts, which will be eliminated by the FPGA, is photon interaction timing. This function determines when photons interact with the detector so coincidence can be determined. As will be discussed in chapter 7, the FPGA in our system processes 64 channels from the photodetectors. A consequence of having so many channels is that the cost and board space required to have a timing ASIC for each of the channels would be prohibitive. This has led us to develop an algorithm to perform the complete timing inside the FPGA without the need for any external components such as analog triggers.

In Chapter 2, the notion of photon coincidence was presented as the way to pair up two photons from a single decay event inside the patient. For the scanner architecture discussed in Chapter 7, the FPGAs for each detector do not directly communicate with each other to determine coincidence. The solution instead is to place a precise timestamp (<3ns) on the arrival time of each photon interaction. These timestamps are sent to the host computer, along with the rest of the pulse information, where the photons inside the field of view (FOV) are paired up based on these timestamps. The accuracy of these timestamps is a crucial piece in the production of high-quality PET images. This is because the timing resolution is directly correlated to the number of non-coincidental events that are accepted as good events, and thus add to the noise of the final image. Recall that the decay events inside the patient occur as a random process so based on the count rate, there will always be some random coincidental pairs. These are photon pairs that are assigned to one single decay event, when in fact they came from separate events, but satisfied the coincidence requirements of the scanner (FOV and timing) as illustrated in Figure 4.1. The exact relationship between the rate of random coincidences and the timing is given by

$$r_{rc} = r^2 t \qquad (5.1)$$

Here, r is the count rate (how many events are interacting with the scanner in the FOV), and t is the maximum difference in time that two timestamps can have in order for them to be paired up as a coincidental pair. This time window is set by the user and is dependent on the precision of the calculate timestamps. If the coincidence time window is too narrow for the timestamp resolution of the system, then true coincidental events may be erroneously filtered out. If it is too wide because of poor timing resolution, then more coincidence errors will occur.

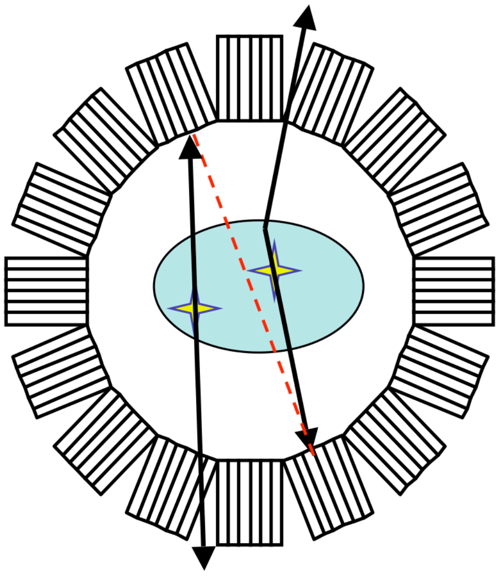


Figure 4.1. Illustration of the source of random chance coincidence. Two decay events occur within the coincidence window, but only one photon from each event interacts with the scanner in the FOV. As a result, the wrong line-of-response is assigned, resulting in additional noise in the final image.

There are many features of the scanner design and timing algorithm that affect the timing resolution of a PET scanner. On the scanner design side, the properties of the scintillator, photodetector and electronics determine the lower limit of achievable timing resolution. When it comes to approaching the lower bound, the design, implementation and features of the timing algorithm are crucial. The properties of the scanner components that make precise timing difficult are noise, variations in pulse shape and amplitude, and variable propagation delays in the system. Since the task of placing timestamps on pulses is attempting to determine when a particular feature of the pulse occurs (start of pulse, certain threshold, peak, etc.), any noise in the signal will introduce jitter into that point. Figure 4.2 illustrates this error for an algorithm that is based on the time that the pulse crosses a certain threshold.

The noise in the pulse means that the crossing of the threshold will have a certain range of time, known as time jitter.

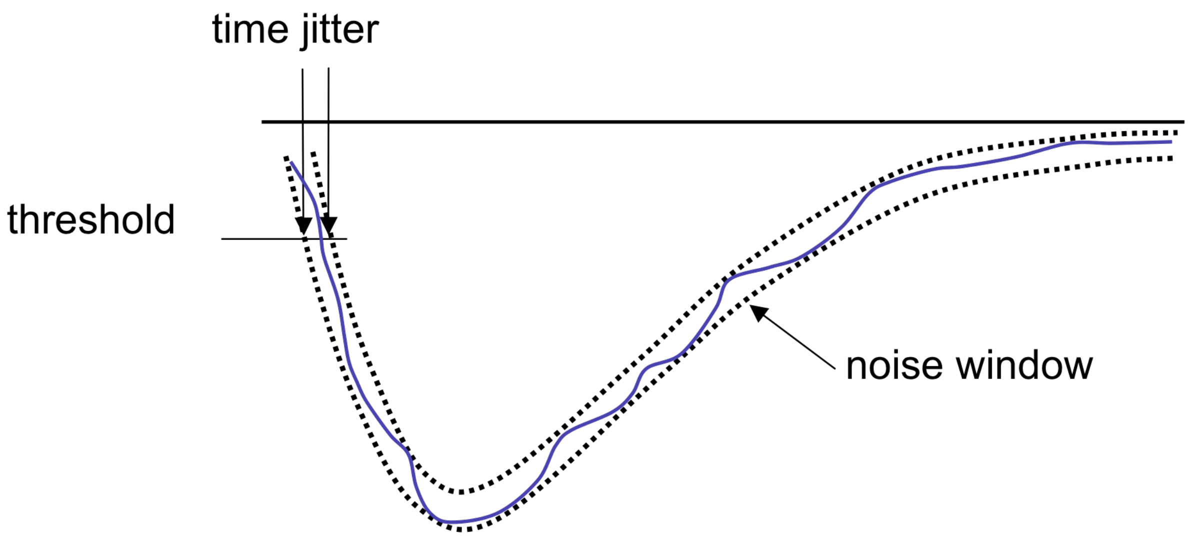


Figure 4.2. Time jitter from noise for a threshold based time stamping algorithm.

Another source of timing errors comes from inconsistencies in the pulse shape and amplitude. The degree of variability is a function of the scintillator and photodetector that generate the pulse. Any difference in the pulse shape will cause similar time jitter to those illustrated in Figure 4.2. Variations in the slope of the leading edge will change the time at which the pulse crosses the threshold.

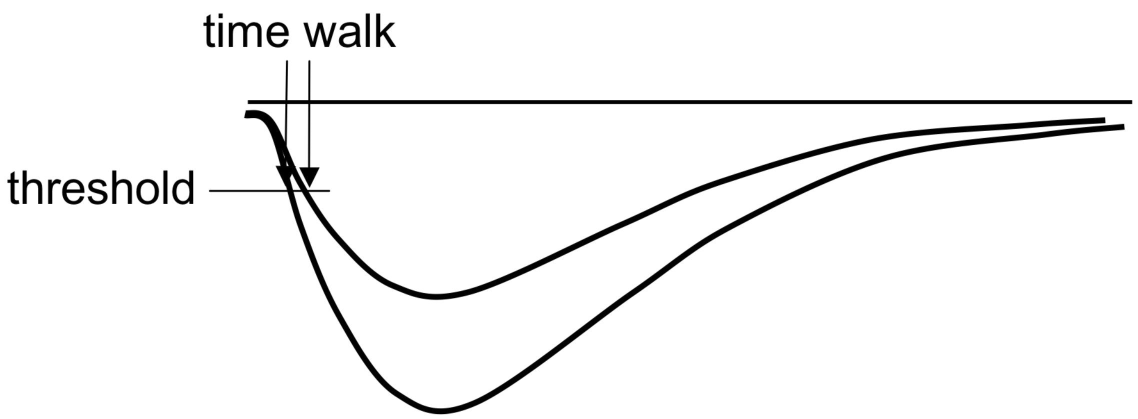


Figure 4.3. Time stamp errors from time walk associated with variation in amplitudes of scintillator pulses.

Fortunately, the photodetectors and scintillators commonly used in PET have a fairly uniform shape. However, they don't have uniform amplitudes. Variation in amplitudes introduces a phenomenon know as time walk (Figure 4.3).

The final source of timing error is variable propagation delays from the photon interaction with the scintillator to the arrival of the pulse at the timing circuit. One large contributor is the position of interaction in the scintillator. If the interaction is on the end of the crystal near the photodetector, then the light will reach the photodetector immediately. However, if it is at the other end, the visible light will have to travel the length of the scintillator before it reaches the photodetector. This error is demonstrated in [13], where the timing resolution is 190ps for a 5mm crystal but increases to 240ps with a 22mm long crystal. The degradation is due to the difference in propagation delay of the light in the crystal.

The figure of merit for timing pick-off is the distribution of the coincidental times stamps. In other words, for a setup with two detectors and a point source exactly centered between them (so coincidental photons arrive at both detectors at the same time), what is the distribution of differences between the time stamps for each detector? For an ideal system, the difference between the time stamps would be zero, but because of the issues discussed above, the distribution will resemble a Gaussian distribution (Figure 4.4). Literature generally reports the full width at half maximum (FWHM) of the distribution histogram. That is, the width of the distribution at half of the maximum count. So, for Figure 4.4, since the max counts is about 6000, the FWHM is about 3ns (-1.5ns to 1.5ns).

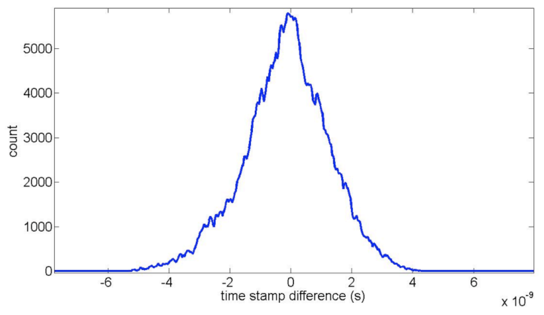


Figure 4.4. An example distribution of difference of time stamps between two pulses for a sampling rate of 70MHz.

4.1: Previous Work

In order to minimize errors, the design of the time stamp algorithms needs to take into consideration the issues discussed above. The two traditional techniques for timing pick-off are leading edge discriminators (LED) [16] and constant fraction discriminators (CFD) [6]. Leading edge is simply determining when the pulse has crossed a certain fixed threshold voltage. This requires an analog circuit that detects the crossing. The drawback of this technique is it suffers from time walk, and it gets worse as the trigger level is set higher. Since the threshold must be well above the noise margins to avoid false triggers caused by noise, leading edge discriminators suffers greatly from time walk.

Current state of the art timing pick-off for PET systems is performed with analog CFDs because they are immune to pulse amplitude variance. A CFD implements a circuit for equation 3.

$$h(t) = \delta(t - D) - CF \cdot \delta(t) \qquad (3)$$

Here, δ(t) is the incoming pulse. The CFD splits the analog pulse into two copies, and delays one copy by D. The other copy is inverted and attenuated by the constant CF (typically ~0.3). Finally, the two altered copies are added to produce a pulse with a zero crossing that can be detected and time stamped. This is shown in Figure 4.5. The zero crossing occurs at a constant fraction of the pulse amplitude for pulses with constant shape.

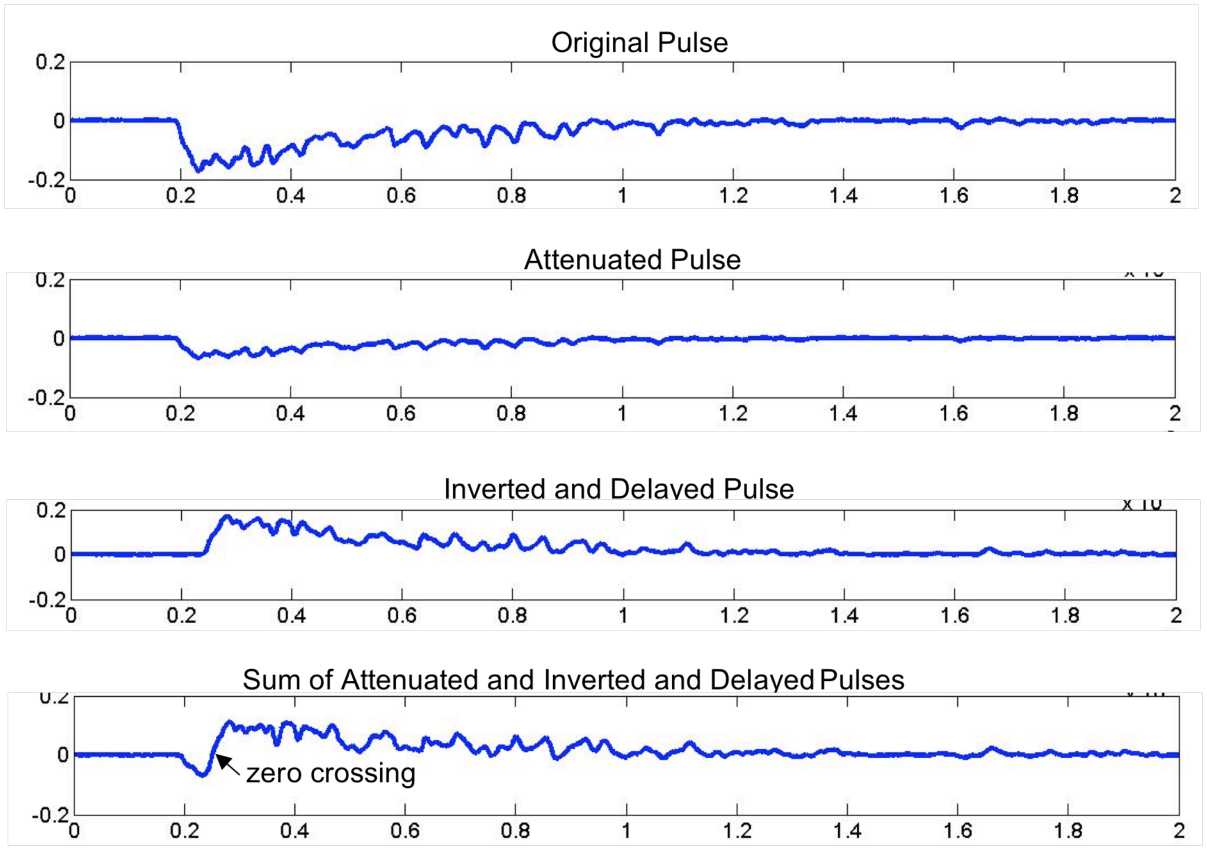


Figure 4.5. Steps in a constant fraction discriminator (CFD). The original analog pulse is split. One copy is attenuated, while the other is inverted and delayed. The sum of these two copies creates a zero crossing that is a good point for pulse timing.

Both CFD and LED must be done in the analog domain (generally in dedicated ASICs), and require a circuit to convert the trigger to a time stamp. While CFDs can achieve sub-nanosecond timing resolution [6], FPGAs have made advancements in computing power and I/O sophistication that may allow them to achieve similar timing results. The challenge of time-stamping pulses with digital circuits is that the ADC may not capture the point that the circuit is looking for. For example, if the circuit is placing time stamps on when a pulse crosses a threshold, the ADC may sample well before and after the threshold is crossed. The lower the ADC rate, the worse this problem is. A solution to this is using an interpolation technique to deduce the point of interest.

There have been previous efforts to perform the timing pickoff in FPGAs. One way is to utilize the increasing clock frequencies to perform a time-to-digital conversion [2]. This method still requires an analog comparator to produce a trigger, and may be limited by the complexity of using fast clocks on FPGAs. Also, FPGAs are gaining in computational power much faster than they are in clock speed, so future scalability is not guaranteed.

Another approach is to utilize digital signal processing to achieve timing precision below the sampling interval. The simplest method is a linear interpolation using two points on the leading edge of the pulse [1,17]. Linear interpolation generally uses the first two points on the leading edge of the pulse to create a line that intersects the two points. This line is then interpolated to the zero crossing or a threshold to determine the start time of the pulse. The problem with linear interpolation is that the leading edge of the curve is not linear, so the farther up the leading edge the two points are, the lesser the slope of the line is (illustrated in Figure 4.6). This, in addition to the noise in the signal, creates a large distribution of starting points.

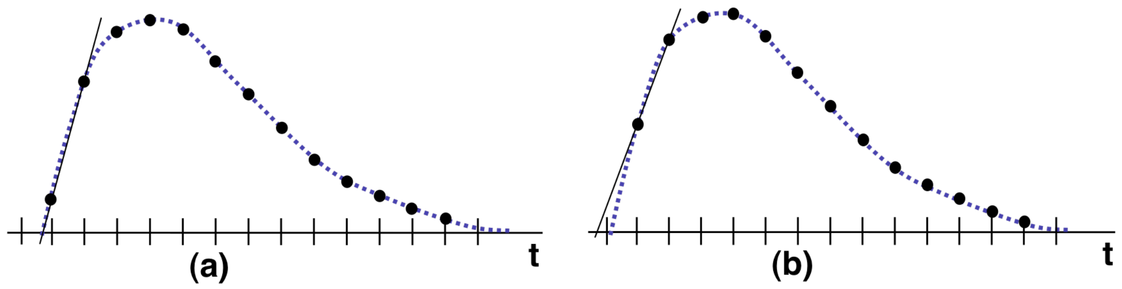


Figure 4.6. Illustration of linear interpolation for pulse start timing. (a) When the first point is near zero, the interpolation is close to the truth. (b) When the first point is farther up the leading edge, the interpolation is far off of truth.

A second all-digital method is the digital equivalent of the CFD shown in Figure 4.5, called digital CFD (DCFD) [17,18]. Since the zero crossing is not necessarily represented by one of the points, the zero crossing is determined by a linear interpolation with the two points on either side of the zero crossing.

Another approach to digital timing uses a matched filter to correlate a reference pulse with the sampled pulse [18]. The time stamp is determined where the two signals have the highest correlation. The filter is normally formulate as a FIR filter:

$$h[n] = \frac{A}{\tau_F - \tau_R}\left(\exp^{\frac{n*T_s}{\tau_F}} - \exp^{\frac{n*T_s}{\tau_R}}\right) \tag{3}$$

A is the reference pulse amplitude, T_s is the sampling period, and τ_R and τ_F are the rise and fall times respectively. The max correlation is determined by finding the max of the convolution of h[n] and the sampled pulse. One issue with this algorithm is that all points on the pulse are assumed to have equal timing quality. We will show in our work that this is not the case. Additionally, the computation for this algorithm requires too much FPGA resources to be a practical solution.

4.2: Algorithm Development

In this work, we set out to design an all-digital timing technique that attempts to mitigate the time jitter associated with noise by using multiple points, as well as reduce the time walk by normalizing the amplitudes of all of the pulses. In addition to these goals, the overall specifications that this method should meet to make it well suited to our PET scanner hardware are:

1) no external circuits besides ADCs (minimizes component count on printed circuit board).
2) work well with lower resolution ADCs, and continue to improve with faster ADCs (minimizes ADC costs and power consumption).
3) does not require too many FPGA resources so multiple instances and other required circuits can co-exist on same FPGA.
4) operates in real-time so important data is not lost.
5) well suited to FPGA strengths.

Using the known characteristics of pulses to compute the start of the pulse, as was done with the correlation algorithm, is a promising method for achieving sub-sampling timing resolution. The method proposed in this chapter uses a reference pulse that is defined at a very high resolution (about every 60ps) and has the same shape as the pulses from the ADC. This reference pulse is "fit" to the scintillation pulse from the ADC and used to interpret the start time of the pulse (Figure 4.7). We assume that the rise and fall times (rise refers to the leading edge or first part of the pulse, and fall is the second part that decays back to zero, even though the "rise" on our crystals is a drop in voltage) of the photodetector pulses are constants, and the variability in the pulses is from the pulse amplitude and white noise. For typical PET scintillators, the rise time is dominated by the response of the photodetectors, while the decay time is a function of the scintillation crystal. Based on these assumptions, the start time of the pulse can be determined by fitting an ideal pulse to the sampled pulse and using the ideal pulse to interpolate the starting point of the pulse.

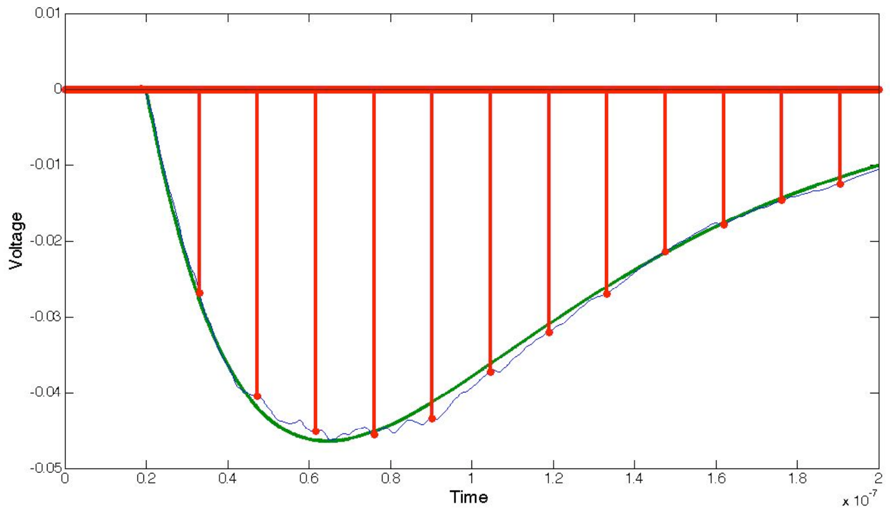


Figure 4.7: An example of a filtered and sampled pulse from a PMT coupled to an LSO scintillator, overlaid with the best least squares fit of a curve.

In order to develop a timing algorithm, we used a 25Gs/s oscilloscope to sample 19 pulses from a PMT that was coupled to a LSO crystal. A 511 keV (^{22}Na) point source was used to generate the pulses. The data from the oscilloscope was then imported into MATLAB. We have chosen to start with simulations in MATLAB because simulations allow the low-pass filter, ADC sampling rate, and fitting algorithms to be quickly investigated before we commit to an implementation.

We hypothesized that if for each incoming event pulse, we created a pulse with two exponentials (one for the rising edge and one for the falling edge) and found the amplitude, time shift, decaying exponential and rising exponentials that produced the best least squares fit, we could use that ideal pulse to interpolate the starting point of the pulse. Using this "brute force" method, the FWHM of the timing pick-off was 2.3ns with a 70MHz ADC. While this is good timing resolution, the search space is far too large for an FPGA to compute in real time. From the "brute force" method, we found that the rise time ranged from .1-.5ns, the decay times ranged from 28-38ns, and the amplitude ranged from .082-.185V. To cover these ranges for a reasonable time step (~60ps) would require the least squares fit to be calculated and compared at least 215,000 times for each pulse (11 decay time steps, 5 rise time steps, 11 amplitude steps and 357 time steps).

Obviously, we must make some compromises to create an efficient real-time FPGA-based algorithm. The first compromise was to use fixed rise and fall time constants. The rise and decay times that gave the best overall least squares fit for all unfiltered, unsampled data were 310ps and 34.5ns. With fixed rise and fall times, the "brute force" method timing resolution degrades to 2.5ns. However, even after eliminating the time constant searches, almost 4,000 searches would still be required for each event.

To eliminate the search for amplitude, we discovered that there is a direct correlation between the area and the amplitude of the pulse, as shown in Figure 4.8. The function to convert area-to-amplitude was determined by sampling each of the 19 pulses with many different starting points, and correlating the area obtained for each sampling to the known amplitude for that complete pulse. Using this estimation, the timing resolution is degraded to 2.8ns.

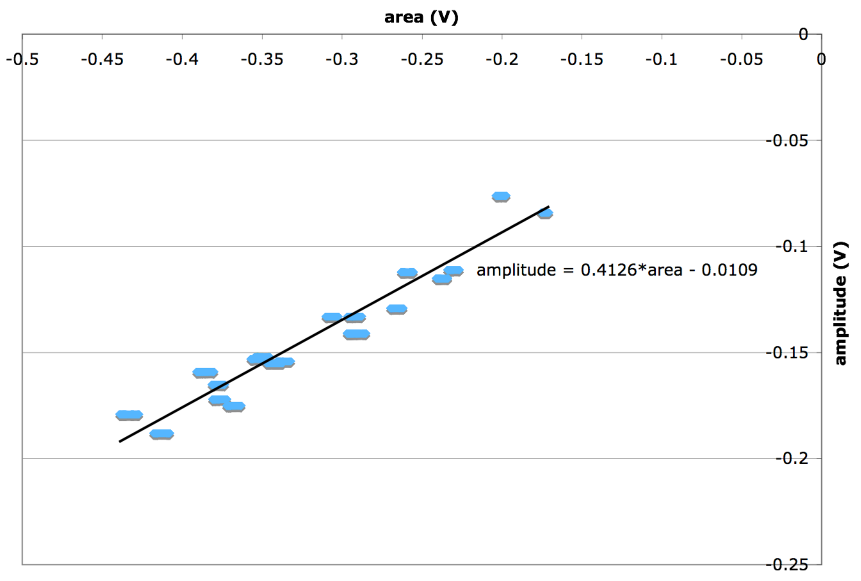


Figure 4.8. Plot of the area of sampled and filtered pulses versus the amplitude of the pulses. The best linear fit is used to estimate the pulse amplitude from the area of the pulse.

Most dimensions of the brute-force search have been eliminated with a loss of only 20% timing resolution, but this would still require 357 searches for all possible timing offsets. However, given that we are fitting the data to a reference curve with known rise and fall times, and computed amplitude, we can convert the brute force time search to a reverse-lookup. We pre-calculate for each possible input voltage the time it occurs on the reference pulse. Thus, each incoming voltage can be converted to a timing offset (i.e. how far is it from the start of the pulse) with a simple memory operation as shown in Figure 4.9. This is done for each sample on the pulse, so that after looking up the 13 points (the length of a pulse with a 70MHz sampling rate) they each have a time at which it "thinks" the pulse started. Unfortunately, if these times are averaged, the timing resolution degrades significantly to 6.5ns.

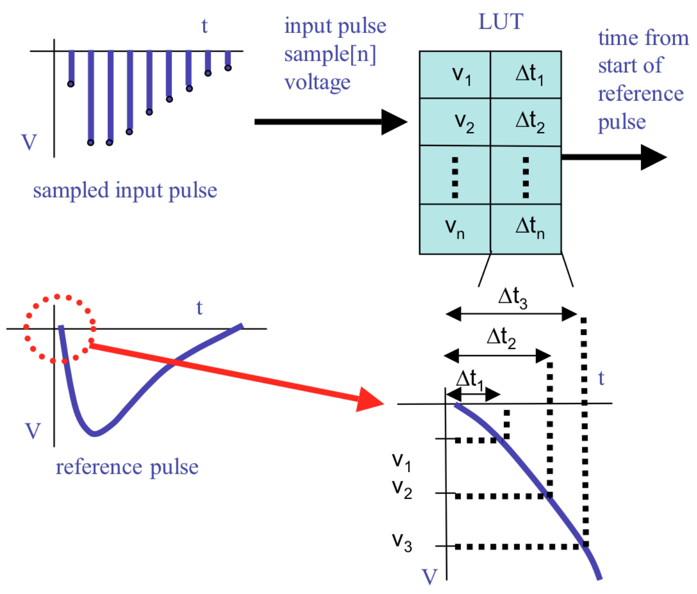


Figure 4.9. Voltage to time lookup method employed to eliminate searching for start time.

After a close inspection of the results from our lookup method, it became apparent that some of the sample points give much better results than others. This is shown in Figure 4.10, which plots the standard deviation of each of the 13 samples. Notice that the deviation is correlated with the slope of filtered pulse, and distance from the pulse start. Points at the peak (points 4 and 5) have a low slope, and thus a small change in voltage results in a large time shift. The tail of the pulse also has a large deviation for the same reason. If only the first point is used, the FWHM is 2.4ns, which essentially equals the “brute force” method. The reason our final algorithm can match the brute force method even with the compromises is because brute force aligns to all samples of a pulse and, as Figure 4.10 indicates, some samples provide better timing information than others.

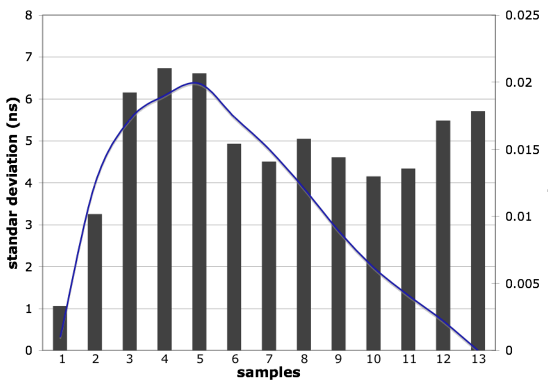


Figure 4.10. Plot of the standard deviation of the points of a filtered pulse that is sampled with a 70MHz ADC. The line is a filtered pulse (also inverted), to give a reference for each point's position on the pulse.

Looking at the unfiltered PMT pulse in Figure 4.11, the timing characteristics of different parts of the pulse makes sense since the rising edge of the unfiltered pulse has much less noise than the rest of the pulse. Using this information, the look-up was changed to only use the first point above .005V on the pulse for a 70MHz sampling rate. This is similar to a leading-edge detector, but automatically eliminates the effects of amplitude variation, and the time is based in the actual voltage of the sample and not the threshold. It also has better noise immunity, since it can test very close to the signal start (where results are most accurate), while eliminating false positives by referencing back only from strong peaks. Additionally, as we will show in the next section, multiple points on the leading edge can be utilized for timing at higher ADC sampling rates in order to further reduce time jitter due to noise.

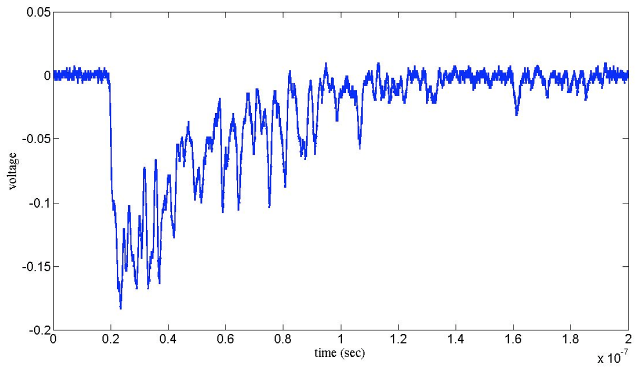


Figure 4.11. Example of an unfiltered pulse from a PMT.

4.3: Testing and Results

To thoroughly investigate this new timing algorithm, a new set of pulses was collected with the 25Gsps oscilloscope from a Zecotech MAPDN (manufactured in 2009) with a LFS-3 scintillator. One thousand pulses from four different pixels were collected. This MAPD has 64 different pixels, and pulses were taken from pixel 5, 22, 47 and 53. The pulses were captured by placing a single LFS crystal on the pixel of interest. The crystal was then moved to the next pixel for the next data set. This larger data set will make the results more statistically sound as well as investigate our algorithm on the emerging solid-state photomultiplier technology. In addition to testing with a larger data set, the reference pulse was changed from a mathematical model to a model that was built from the actual pulses. The reference pulse for each pixel is an average of all of the pulses in that data set after they were aligned and normalized for amplitude. The reason for this change was twofold. First, a pulse built from actual pulses is a slightly better fit than a reference pulse built with a mathematical mode. More importantly, as we will present in the next chapter, it is easier to form the reference pulse from real data than to determine the correct mathematical model.

Again, this data set was imported into MATLAB to simulate the following parameters

1) low-pass filter cutoff frequencies as shown in Figure 4.12 (5MHz, 10MHz, 20MHz, 40MHz, 100MHz, 200MHz and 500MHz)
2) number of points used on leading edge for timing lookup
3) ADC sampling rates (65MHz, 125MHz, 300MHz, 500MHz and 1GHz)

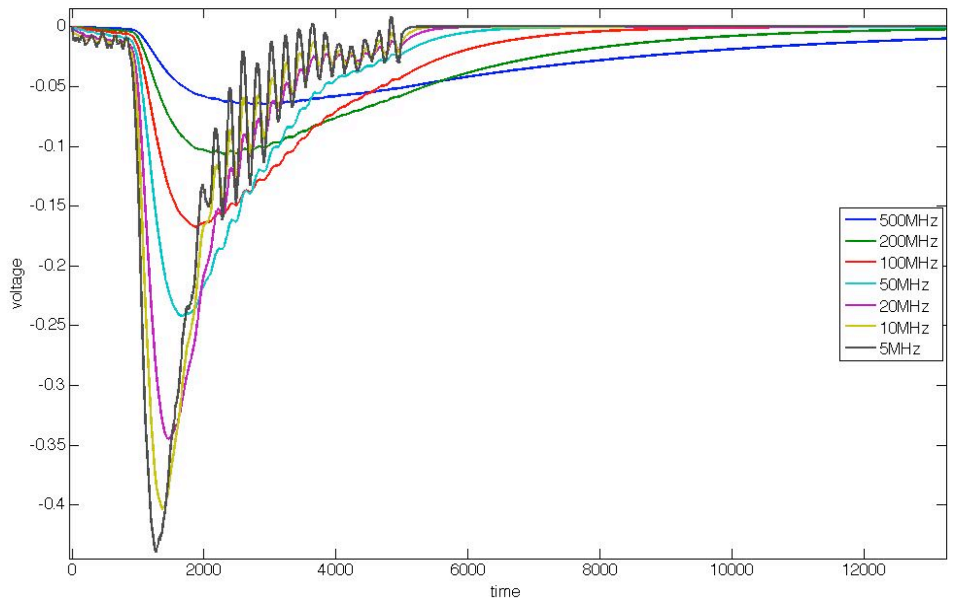


Figure 4.12. Illustration of the pulse shape for the different low-pass filter cutoff frequencies.

These parameters need to be investigated simultaneously because of their interrelationship. Generally, the steeper the leading edge, the better the timing resolution will be. However, a steep leading edge will have fewer samples on it, and using multiple samples for timing may reduce the time jitter. Thus there is a tradeoff between getting more samples on the leading edge with a lower frequency cutoff and producing a steeper edge with a higher frequency cutoff. Since these two parameters (number of samples available on the leading edge and the cutoff frequency) are opposed, a study of the right balance was performed. To test this, the coincidence timing was checked for the data sets from two different pixels (5 and 22). To create coincidental data sets, the pulses from each pixel were aligned in two pulse streams. That is, the single pulses were placed back-to-back to simulate a stream of pulses from the ADC so that each pulse in one set aligned with a single pulse in the other set. The timestamps from the pulses in the two streams were compared to produce a distribution. For each sampling rate, all appropriate cutoff filters and number of samples were investigated. For lower sampling rates, the pulse has to be slowed down enough (lower frequency cutoff) to assure at least two samples were on the leading edge. This means that the higher frequency cutoffs need not be investigated for lower ADC sampling rates. Likewise, only a certain set of the number of different sample used in timing merits investigation for each ADC sampling rate. For example, for 65MHz, it only makes sense to use one sample on the leading edge because the second sample is often near the peak of the pulse. For 1GHz, utilizing 5-10 samples was tested. Below 5 samples doesn't make sense as the results improve above 5 samples so it can be assumed that below 5 samples would produce worse results. This has been verified with a few spot checks. The results for these tests are shown in Table 4.1. The best results for each ADC sampling rate are in bold. The best points are a mix of a steep rising edge while still being able to get multiple samples on the leading edge. Also, whenever there were equal or even close results, the number of samples that is a power of two was chosen for ease of FPGA implementation.

Table 4.1. Timing FWHM (ns) for different filter cutoffs and samples used for the different ADC sampling rates.

		lowpass filter cutoff (MHz)						
ADC sampling rate	samples used	5	10	20	40	100	200	500
65MHz ADC	1	5.01	**3.11**	--	--	--	--	--
125MHz ADC	1	4.28	1.63	1.63	1.31	--	--	--
	2	4.26	2.09	1.98	**1.27**	--	--	--
	3	4.35	2.46	2.19	1.70	--	--	--
300MHz	1	4.03	1.52	1.08	1.04	1.38	1.13	--
	2	3.91	1.70	1.15	0.92	1.10	**0.83**	--
	3	4.12	1.86	1.29	0.92	1.06	0.83	--
	4	4.16	2.00	1.45	1.04	1.15	1.04	--
500MHz ADC	2	3.80	1.68	1.04	1.04	0.81	0.71	0.81
	3	3.77	1.75	1.10	1.04	0.76	0.67	0.69
	4	3.80	1.82	1.17	1.01	0.76	**0.67**	0.74
	5	3.77	1.91	1.27	1.04	0.81	0.74	0.87
	6	3.86	2.00	1.33	1.08	0.87	0.85	1.08
	7	3.89	2.12	1.43	1.15	0.99	1.04	1.38
1GHz	5	3.75	1.93	1.08	1.06	0.85	0.76	0.69
	6	3.70	2.00	1.17	1.06	0.83	0.71	0.67
	7	3.73	2.00	1.20	1.04	0.81	0.71	0.67
	8	3.80	2.07	1.24	1.04	0.81	0.71	**0.67**
	9	3.80	2.12	1.29	1.06	0.81	0.74	0.81
	10	3.82	2.14	1.33	1.08	0.83	0.76	0.90

”--“ – indicates the pulse is too short to reliably get samples on the leading edge.

For the best combination of filter cutoff and samples used, the timing resolution for all possible combinations of the four pixels was calculated for each ADC sampling rate. These results are presented in Table 4.2. Even though the test for the 5_22 line in Table 4.2 is the same as the highlighted tests in Table 4.1 there are slight differences. This is because the pulses were randomly sampled to simulate a free running ADC.

Table 4.2. Timing resolution FWHM (ns) for different ADC sampling rates based on best parameters in Table 4.1.

	ADC sampling rate				
pixel pairs	65MHz	125MHz	300MHz	500MHz	1GHz
5_22	3.11	1.29	0.83	0.71	0.74
5_47	2.85	1.13	0.76	0.62	0.67
5_53	2.99	0.97	0.74	0.67	0.69
22_47	3.80	1.45	0.92	0.74	0.81
22_53	3.15	1.33	0.87	0.85	0.76
47_53	3.06	1.38	0.85	1.04	0.78
average	3.16	1.26	0.83	0.77	0.74

Gauging the quality of timing resolution for a given algorithm is difficult because it is dependent on the intrinsic properties of the detector and electronics as discussed above. Essentially, there is a lower bound for the timing resolution that is achievable given a particular detector setup. Also, because these pulses were collected as singles and manually aligned into two simulated coincidental data stream, the quality by which they were aligned plays a factor as well. In order to perform a fair comparison, the same data should be run through a known algorithm as a reference. In order to perform this comparison, as well as see how this all-digital algorithm compares to an analog method, the above data set was also run through a constant fraction discriminator (CFD) and a leading edge discriminator (LED). Two variations of the LED were simulated, one with amplitude normalization where the pulse peaks were all normalized, and one where the pulses were not normalized. In a real implementation, amplitude normalization would require an additional analog circuit. In order to simulate the analog versions on the data set, the pulses were not down sampled from 25Gsps as they were in the all-digital simulations. Even though this is still technically a digital simulation, we feel that the sampling rate is high enough to imitate a continuous-time algorithm. The best attenuation factor (.3) and delay factor (250ns) for the CFD, and trigger (.37V) for the LED was determined. These results, along with the average timing resolution from Table 4.2, are shown in Figure 4.13.

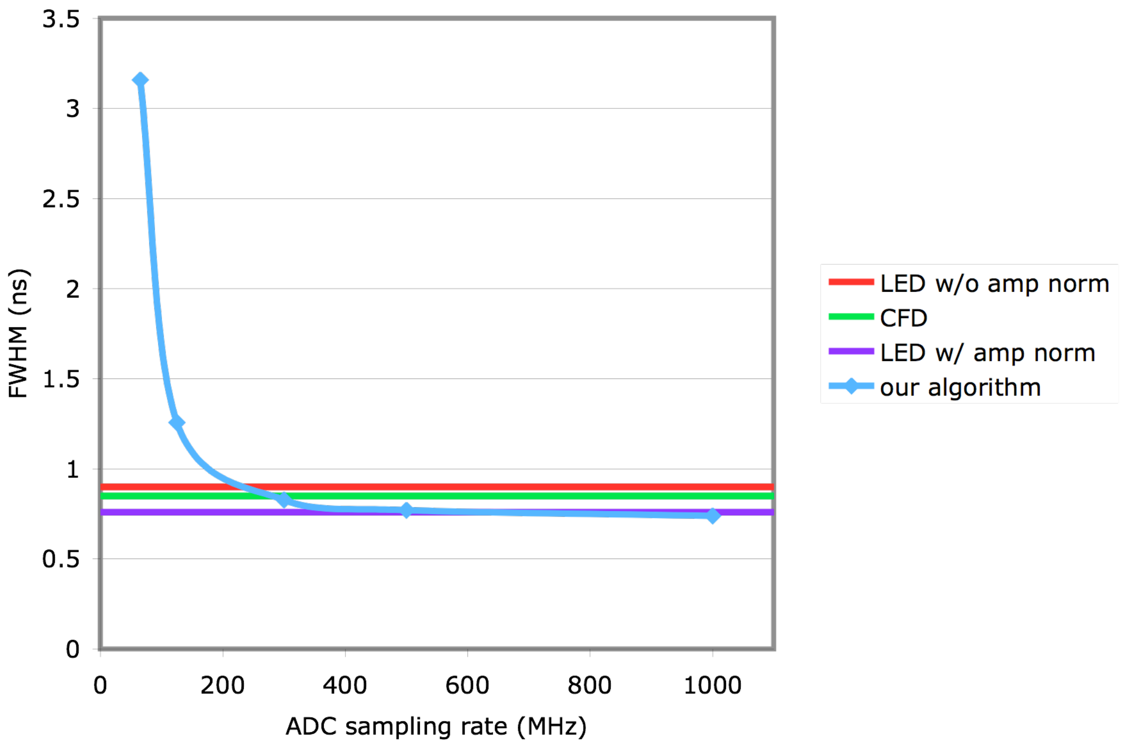


Figure 4.13. Timing resolution FWHM (ns) for a simulation of our all-digital algorithm and a leading edge detector (LED) without amplitude normalization, LED with amplitude normalization and constant fraction discriminator (CFD).

Notice that our algorithm outperforms the CFD and LED without pulse normalization at about 300MHz, and equals the LED with normalization at about 500MHz. It is also within 9% of the LED with amplitude normalization at a 300MHz ADC sampling rate. The reason it outperforms the CFD is because the constant fraction that the zero crossing represents has to be far enough up the leading edge to assure a reliable initial dip before the zero crossing in the presence of noise. The LED without normalization suffers from time walk as discussed above. The reason our algorithm can almost match the LED with normalization at 300MHz is because we utilized multiple points on the leading edge to find the start time of the pulse, while the analog version uses only one threshold. In theory, an analog version could have multiple thresholds, but that would greatly increase the complexity of the circuit.

4.4: Implementation

One of the goals in the development of this algorithm was to produce a circuit that has an efficient FPGA implementation. With this consideration, as well as the results above, the algorithm is as follows:

1) remove the baseline from the incoming ADC stream.
2) keep a running sum of the previous N ADC samples (where N is the length of a pulse).
3) detect the start of a pulse using a trigger.
4) check if the energy of the pulse (summation of its samples) is within the limits of a “normal” pulse.
5) normalize the first M samples of the pulse, based on the ratio of reference pulse summation to current pulse summation (M is the number of samples used for timing at a the particular ADC sampling rate, see Table 4.1).
6) look up normalized sample(s) to determine how far they are from the start of the pulse.
7) calculate average of time lookups from step 6.

The architecture for this algorithm is shown in Figure 4.14. The box labeled rc signifies a low-pass filter.

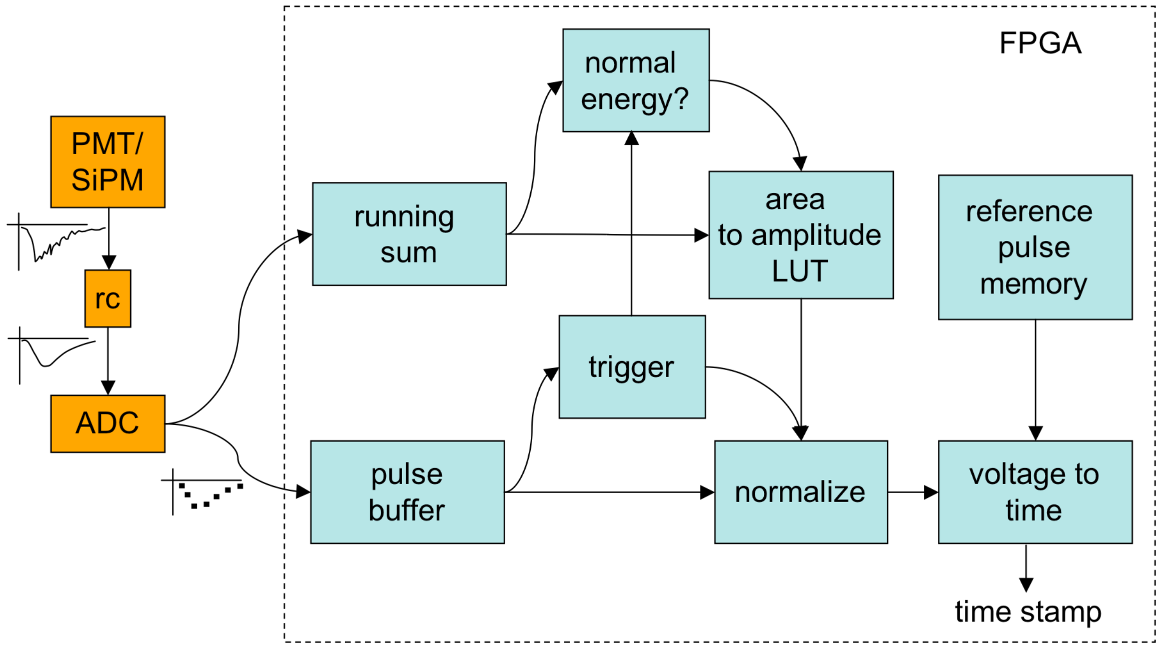


Figure 4.14. The architecture of the timing pick-off circuit implemented in the FPGA.

4.4.1: 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. That is, even in the absence of event pulses, the signal coming from the photodetector has a small voltage offset in addition to noise. This baseline can also 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 blindly averaging a small number of samples before the pulse may give incorrect results as well. Given these constraints, any baseline restore algorithm should fulfill two requirements:

1) able to react to baseline moves but not high-frequency noise.
2) not be affected by pulses.

We have developed an algorithm that accurately calculates the baseline and adheres to these requirements. In this algorithm, a new baseline value is calculated each ADC cycle via a running average of the last 8 samples of baseline. This allows the baseline to move, but smoothes out the higher frequency noise. Also, by using the average of eight samples, the average is simply calculated by right shifting by three.

To keep the baseline from drifting due to data from events, as each sample is read into the FPGA, we 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. The sides of the window are centered around the baseline in order to move with the baseline. To meet these requirements, the baseline window uses two counters to keep track of how many samples have been inside of the window ("IN") and how many have been outside of the window ("OUT"). Let "LIMIT" be the smallest power of two no smaller than three times the pulse length. When "IN" reaches "LIMIT", then "IN" resets and the window contracts by one ADC least significant bit (lsb). Likewise, when "OUT" reaches "LIMIT", then "OUT" resets and the window expands by one ACD lsb. The count is defined in terms of pulse lengths because different setups will produce different pulse lengths, and the window should only shift after the longest expected span of pulses. As chapter 6 will present, pulses can pileup in a succession of overlapping events. In our system, a 4-pulse pileup event will be very rare, so if the baseline is outside the window for more than three times the pulse length, it is most likely due to a shift in the baseline. For other systems, the length may need to be adjusted. For implementation purposes, the count is set to a power of two that is close to the desired length. The baseline window is initialized to be ±1 (1 significant ADC bit) and allowed to expand during warm-up.

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 (Figure 4.15). 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.

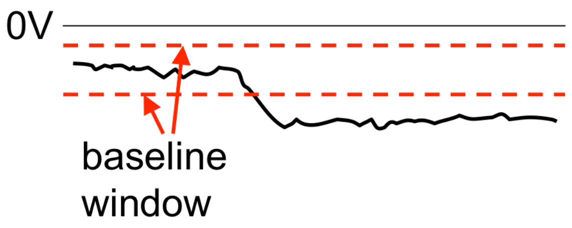


Figure 4.15. Illustration of a baseline shift greater than the baseline window.

4.4.2: Pulse Energy

The pulse energy is an important metric to know about each pulse. The primary reason is to differentiate between "good" and "bad" pulses. When a 511keV photon completely interacts with the detector, the energy of the resulting pulse (summation of all of its samples) will be within a given range. In addition to these "good" pulses, other pulses with varying energies are also produced by the detectors. These "bad" pulses are from various sources such as partial photon interactions and radioactive sources inherent to the scintillators. To filter out the "bad" pulses, energy quantification is used. This is the process of only accepting pulses that fit within a narrow range of energy.

To determine the pulse energy, a running sum keeps the summation of the last N samples from the ADC, which corresponds to the length of a pulse for the given sampling rate and filter cutoff frequency. To accompany the running sum, a buffer delays the samples from the ADC by the same N samples before they are checked to see if they constitute the start of a pulse. In this way, when the trigger detects the start of the pulse, the summation of the pulse is already available to verify that the pulse energy is within the expected range.

4.4.3: Trigger

The trigger that determines the start of a pulse is an important component of this algorithm. The trigger is designed to find the first sample of the pulse that is as close to the baseline as possible. Generally, the closer the sample is to the baseline, without being corrupted by noise, the better the timing results will be. In our system, the trigger inspects two samples on the leading edge. The first sample is tested to determine that it is above a certain voltage threshold. This threshold is set close to the baseline, but generally above the noise in the baseline so that the trigger is not tripped by noise too often. The second sample is checked to determine if it is a certain voltage above the previous sample. The amount of increase between the two samples is dependent on the pulse shape and the ADC sampling rate. The reason the second pulse has to be checked is because the threshold for the first sample is close enough to baseline that it will occasionally be tripped by noise. As illustrated in Figure 4.16, if the trigger relies only on the threshold crossing of one sample and the pulse energy, it is possible that noise could cause the trigger to pick the wrong sample. In this scenario, a noise spike is above the threshold and just before the pulse, so the summation of the next N samples could easily be above the pulse energy minimum, since the samples of the pulse that are not summed are on the tail of the pulse. Our trigger would ignore this sample because the next sample does not meet the criteria, and the first sample will be correctly found three samples later.

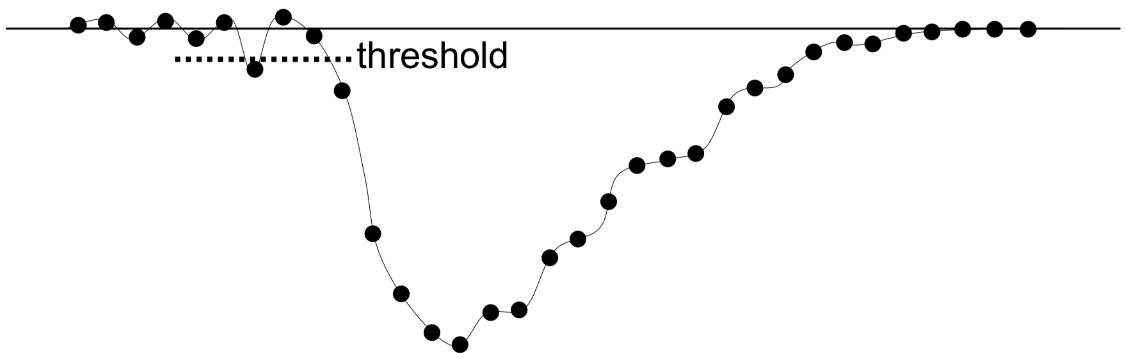


Figure 4.16. Illustration why a two part trigger is needed.

4.4.4: Amplitude Normalization

Once a pulse is detected and verified, it is normalized to the reference pulse to eliminate any time walk. Notice that only the samples to be used in the timestamp lookup need to be normalized. The normalization factor is the ratio of the energy of the reference pulse to the energy of the current pulse being processes. This value is precomputed and stored in a lookup table that is indexed by the energy of the current pulse being processed. Recall that the reference pulse is stored at a much higher resolution, so the energy summation has to be based on a down-sampled version of the pulse to match the sampling rate of the ADC. The samples that will be used in the timestamp lookup are multiplied by the result of this lookup.

4.4.5: Time Lookup

The final step of this timing algorithm is to convert the voltage of the normalized samples on the leading edge of the pulse to a timestamp that indicates where the pulse started. Each timestamp is composed of two parts, a coarse-grain part and a fine-grain part. The coarse-grain time is simply a counter that keeps track of the ADC samples. The fine-grain time is produced by the voltage-to-time lookup and is essentially a division of the coarse-grain time as shown in Figure 4.17. Fundamentally, the whole purpose of this timing algorithm is to determine which of the fine-grain time divisions between the ADC samples is closest to the start of the pulse. The exact step size of the fine-grain timestamp is dependent on the ADC sampling rate. It is designed to be around 60ps, but it should be such that the ADC sampling period is an even power of two multiple of the fine-grain time steps. For example, if the

ADC sampling period was 15ns then the fine-grain resolution would be 58.6ps steps (15ns/256) and there would be 256 fine-grain steps in between the coarse-grain steps. This assures that coarse-grain and fine-grain times are coherent. The coarse-grain portion is required because the coincidental pairs are determined offline by the host computer. Without a coarse-grain time, there would be no way to differentiate between pulses that had similar fine-grain timestamps, but were actually separated in real time by a significant amount.

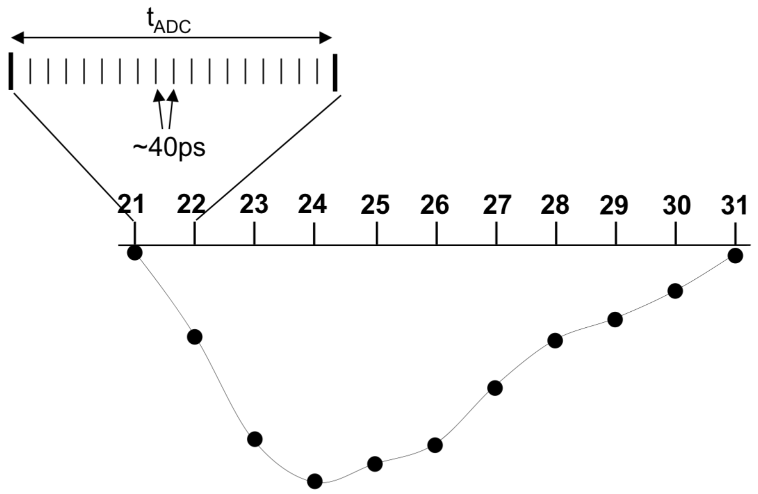


Figure 4.17. Illustration of the coarse-grain and fine-grain portions of the timestamp.

The coarse-grain portion of the timestamp is implemented as a simple counter that increments with every new ADC sample. This counter does not have to count without rolling over for the whole scan. This is because events are sent to the host computer in order of occurrence, so the coarse-grain counter just has to be large enough so an adequate amount of events occur before it rolls over. In our FPGA implementation, a 12-bit counter is more than enough. The fine-grain portion of the timestamp originates from the voltage-to-time lookup. This is a pretty simple lookup table as illustrated in Figure 4.9. Notice that the whole reference pulse does not have to be stored because only samples on the leading edge are utilized. At higher ADC sampling rates, there will be multiple look-ups for each timestamp because multiple samples are used. This will require multiple cycles to achieve, but since only one pulse is processed at a time the latency is not an issue. In other words, the system has to check every sample for the start of a pulse, but once one is detected, the system has the length of a pulse to process the pulse before it has to start looking for the next pulse. The multiple fine-grain timestamps are averaged to calculate the final timestamp. This is accomplished by a simple shift since the number of samples used is a power of two by design. Finally, the coarse-grain and fine-grain portions are combined, where the coarse-grain portion is essentially the integer portion and the fine-grain is the fractional portion of the timestamp. Notice from Figure 4.17 that the fine-grain portion has to be subtracted from the coarse-grain count because the first sample on the pulse is after the start of the pulse, and the fine-grain portion indicates how far back in time the start of the pulse is.

For the host processor to convert these timestamps into coincidental pairs, there needs to be some method to relate the timestamps from two separate FPGAs. That is, when the host computer sees two identical coarse-grain timestamps from two FPGAs, it needs to know for sure that they truly did occur in the same "global" clock phase. Since the coarse-grain timestamp is based on a counter that runs off of the local FPGA clock, the scanner design needs to ensure that all of the counters start at the same time and that the clocks on the separate FGPAs stay in phase through the length of a scan. This is generally done by distributing a single clock signal to all FPGAs and programming a PLL to generate the internal FPGA clock off of this global clock.

4.5: Timing Experiment

To test the timing algorithm in a real system, we ran an experiment with a positron source, PMTs, scintillators, low-pass filter, ADC, and FPGA. Figure 4.18 illustrates the experimental setup. The positron source was placed in between the two detectors, and coincidental events were detected and timed. To determine the timing resolution, the difference between time stamps for paired events was binned into a histogram and measured for FWHM. The FWHM allows for the source to be slightly off center because any difference in the time of flight for the two positrons will be constant and the center of the histogram will be shifted from zero, but the FWHM will not change. One factor to be cautious of in a timing experiment is the count rate of the detectors. If the count rate is too high, there will be a high chance of pulse pileup, which will cause errors in the timing algorithm. To avoid this, the distance between the source and detector was increased until there was a large separation between observed events.

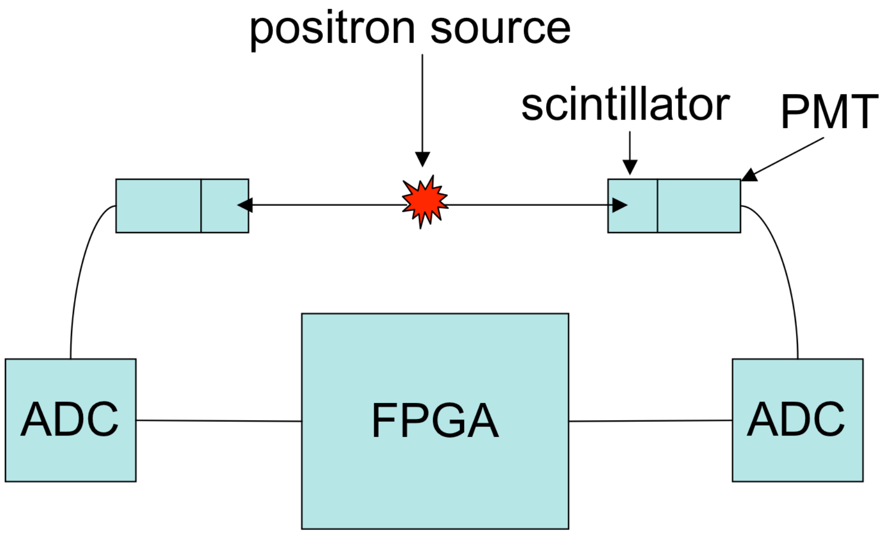


Figure 4.18. Experimental setup to validate the timing algorithm simulations.

This timing algorithm was implemented on a StratixII DSP Development board as shown in Figure 4.19. This board has two 12-bit ADCs (125MHz maximum sampling rate) so the detectors can be set up in coincidence to simulate two detectors on opposite sides of the scanner. A detector was attached to each ADC, which feeds into a timing circuit in the FPGA. The design was placed and routed using Quartus II 8.0. Each timing core uses 250 adaptive logic modules (ALMs) (out of 71,760), 114 kbits of memory (out of 9.4 Mbits), 1 phase-lock loop (PLL) (out of 8), and 36 I/Os (out of 743). As will be discussed in chapter 7, there may need to be more than one timing circuit in the FPGA since there will be multiple channels supported by each FPGA. Each timing circuit could handle multiple channels, so the number of timing circuit instances will depend on the count rate of each channel.

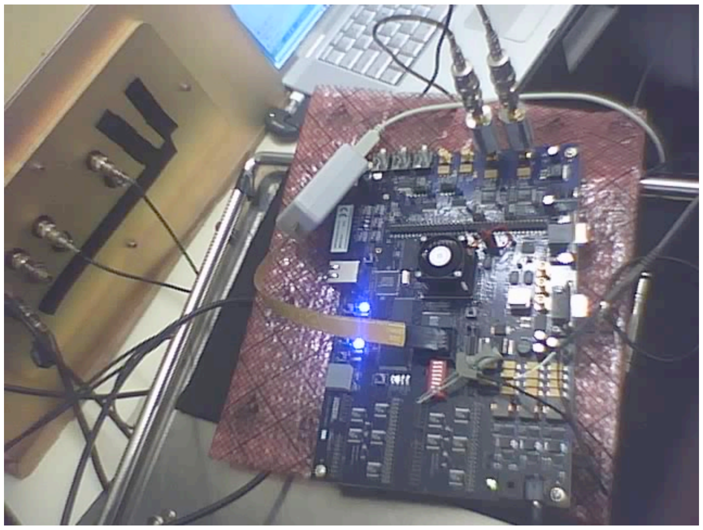


Figure 4.19. Timing experiment setup. The two ADCs on the Altera StratixII DSP development board were connected to two photodetectors that had a radioactive source placed between them inside a light-tight box.

Figure 4.20 gives the results for the timing experiment with the FPGA and real detectors for sampling rates of 60MHz and 125MHz. The same scintillator/detector setup was also tested with a CFD to get a baseline of the intrinsic timing properties of the setup. The CFD is about 3X worse in this setup than in our previous simulations with the MAPDN. This is an indication that this particular setup has poor intrinsic timing resolution. This may be due to an extraordinary amount of noise in the experimental setup or the properties of the particular scintillator and photodetector used. Even given this, the measured timing resolution from the FPGA seemed too high. The hypothesis was that there were two issues that negatively affected the timing results. The first issue was the frequency cutoff of the low-pass filter. In the experimental setup, only a 50MHz low pass filter was available, while in simulation a lower cutoff produced the best timing results. The other issue was with the reference pulse. To build the reference pulse for this experiment, pulses were acquired with a 25Gsps oscilloscope just after the low-pass filter but before the signals entered the printed circuit board. These signals were then imported into MATLAB to determine the reference pulse. The reference pulse was then loaded into the design with Quartus. The problem with this method is the effect that the circuitry from the filter to the FPGA has on the shape of the pulse is not captured. This includes the printed circuit board (PCB) connector, the ADC, the

PCB traces, and the FPGA I/O. This circuitry changes the shape of the pulse so that it no longer matches the shape of the reference pulse.

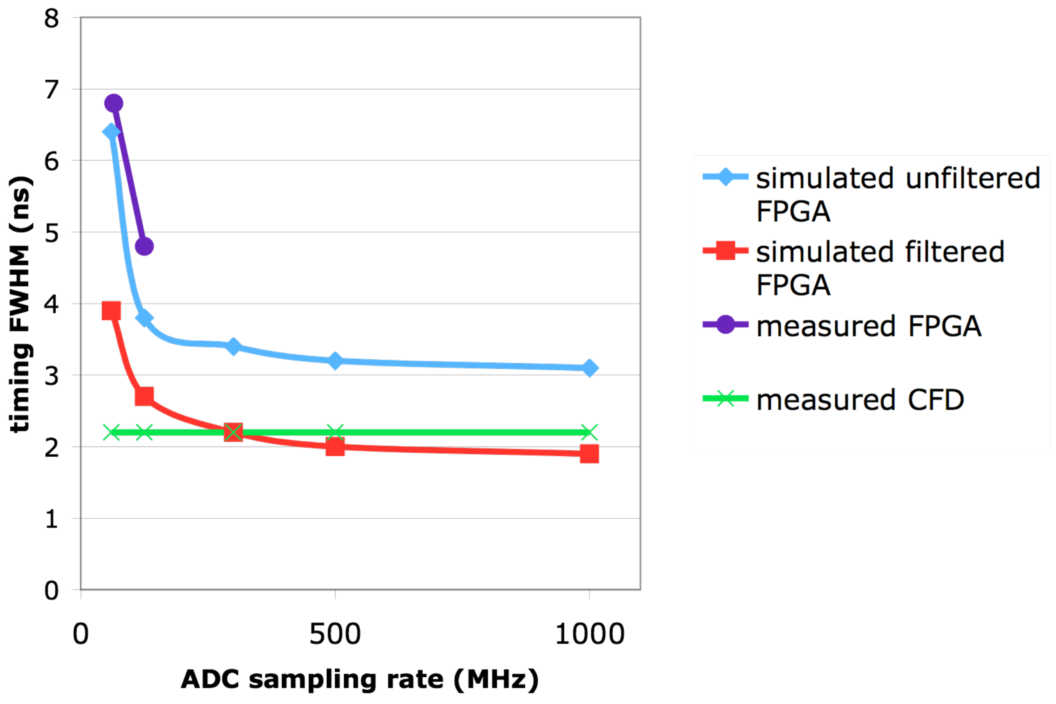


Figure 4.20. Timing results for simulated and experimental coincidental timing. Note that CFDs do not use ADCs, but the data is included to give a reference.

To determine how these issues affect timing, as well as how the algorithm will perform as the ADC technology advances, MATLAB simulation results for our timing algorithm are also included in Figure 4.20. To simulate the timing algorithm, the data that was collected for the reference pulse was also used for a simulation. The data was imported into MATLAB and the timing algorithm was then run on the data for simulated sampling rates from 60MHz to 1GHz. This is reported in Figure 4.20 as simulated unfiltered. This indicates how the mismatched reference pulse affected the timing since the data was also sampled just after the filter. To test the effect of the low-pass filter, the same data was filtered again in MATLAB before timing. The result for the best timing is shown in Figure 4.20 as simulated filtered. Notice that with appropriate filtering, our algorithm will achieve resolution similar to that of the CFD given a fast enough ADC.

4.6: Conclusions and Future Work

This chapter proposes a novel all-digital timing algorithm that is derived from a pulse fitting idea. Traditionally, timing has been a function that was performed with custom analog circuits even in systems that contained an FPGA. Eliminating these circuits and migrating the timing function into the FPGA will eliminate the need for these extra circuits, which will save board space and system power. However, computing the start time of the pulse in the digital domain requires some kind of interpolation, since there is a low probability that the ADC will capture the point of interest. There have been previous efforts in digital event timing for PET, but none fit our criteria of a small footprint, real-time operation, and no external circuits necessary, while still achieving good timing resolution.

The algorithm that we have developed in this chapter fulfills all of the requirements set out for a good FPGA implementation. The starting idea of this algorithm was to fit a reference pulse to incoming photodetector pulses to interpolate the start time of the pulses. While this method achieved good timing results, it was obvious from the beginning that it would not yield a reasonable real-time FPGA implementation. In the process of eliminating the parts of the fitting algorithm that had a search component, we developed a routine that is a hybrid of a leading-edge discriminator and a pulse fitting routine. It was discovered that the leading edge of the pulse had the best information for timing, and the trailing edge had so much noise that is was detrimental to determining the start time of the pulse.

The results from the final algorithm are very promising. It achieves a timing resolution of about 750ps FWHM at a 1GHz sampling rate (this is for an older generation SiPM and should improve as better perfofming SiPMs become available). Interestingly, if the ADC sampling rate is dialed back to 300MHz (the rate of the timing channel in our scanner) the timing resolution only degrades to about 830ps. Additionally, with a 500MHz ADC, it meets the timing resolution of the best theoretical analog timing algorithm. That is an analog circuit that has ideal amplitude normalization, which would be difficult to implement. We were able to match the analog circuit because our two-part trigger ensures that the samples are on the leading edge, and we use multiple samples to perform timing.

The one drawback to this algorithm is the need for accurate parameters of the pulse. Our algorithm requires correct knowledge of the pulse length, voltage steps of the leading edge (for the trigger) and most importantly, the shape of the pulses for the reference pulse. For the development of this algorithm we originally used an exponential based mathematical model of the pulse, but switched to a pulse model based on averaging the actual pulses. A reference pulse built from actual pulses is a much better fit and, as the next chapter will show, it is easier to build in a real system. That said, this is a disadvantage of this algorithm over one that doesn't require accurate knowledge of the pulse shapes, such as the digital constant fraction discriminator.

One possible improvement for this algorithm is the use of variable slopes on the leading edge. In the initial work, it was found that the leading edge had a rise time that varied from 100ps to 500ps. In order to eliminate the search algorithm, we settled on one single rise time and then transitioned to one reference pulse. Now that the algorithm has removed all of these search steps, it may be possible to add the search for multiple slopes on the leading edge to improve the timing results. This only would make sense at higher ADC sampling rates were multiple samples are assured on the leading edge. To accomplish this, multiple reference pulses would need to be stored, and the one that is the best match of an incoming pulse would be used. To determine the match, it may be possible to utilize the separation between the multiple points on the leading edge. Notice that the timestamps on the two successive points should be separated by one ADC sampling period in terms of fine-grain time stamps. If the slope of the leading edge on the reference pulse were steeper than that of the data pulse, then the separation would be less, and vice versus for a leading edge that is too shallow. The difficult portion of this added step would be accurately calculating multiple reference pulses via the algorithm that will be discussed in the next chapter.

Chapter 5 : Pulse Parameter Discovery

The timing algorithm discussed in chapter 4 relies on accurate knowledge of certain parameters of the photodetector pulses, the most important being the shape of the pulses. Unfortunately, our research has indicated that some these parameters can vary between photodetectors (reference pulse shape and pulse length) and over time (event triggers). We also indicated in the previous chapter how difficult it is to obtain an accurate pulse model even with a manual method. In this chapter, we show how we 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, is the need to accurately determine or discover a good reference pulse. Previous timing work [18] used exponentials to model the photodetector pulses. Our experiments found that models based on exponentials did not fit the data as well as models derived from the actual photodetector pulses. Moreover, the inaccuracy is largest at the initial portion of the pulse where the reference pulse is utilized for timing. One reason for this is the inability to accurately model the electronics between the photodetector and the FPGA (Figure 5.1). All components of this chain (amplifier, low-pass filter, ADC, FPGA input pads, printed circuit board) can affect the shape of the final pulse. If a mathematical model is used for the reference pulse, the shaping effect this chain has also needs to be modeled. This can be difficult, and would have to be recalculated if any components were changed. Additionally, we found that for MAPDs, each pixel can produce a different pulse shape. Alternatively, the FPGA can automate a process that would be very laborious for a full scanner. All of these factors indicate that a solution where the FPGA builds reference pulses using sampled data may produce the best possible timing results.

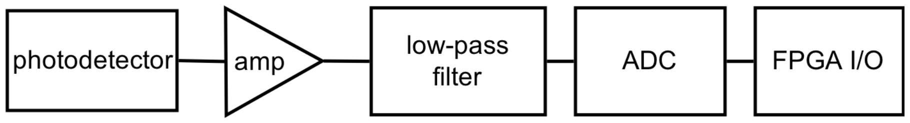


Figure 5.1. Typical data acquisition chain for a PET scanner.

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

5.1: Incorrect Reference Pulse

Before the algorithm was developed, we verified that poor timing resolution results for the FPGA implementation in chapter 4 were due in part to a mismatched reference pulse. This was accomplished by running a timing simulation using the wrong reference pulse. The same simulation that was done in chapter 4 was performed again, but the reference pulse used for amplitude normalization and timing lookup was a reference pulse based on data from another pixel. This simulates the cost of using a single reference pulse for the whole scanner. The results from this simulation are presented in Figure 5.2.

At a 65MHz ADC sampling rate, the average timing resolution is 140% worse if the wrong reference pulse is used. If the ADC rate is increased to 125MHz the difference is down to 56%, and it is about 30% worse for the rest of the sampling rates. The reason the timing resolution with the wrong reference is especially bad at lower sampling rates is because more of the reference pulse is used for timing with a faster sampling period. At 65MHz, the average voltage of the first sample is about 45% of the pulse peak after normalization. For the rest of the ADC sampling rates, the average voltage is about 25% of the peak value. For ADC rates that use multiple samples for timing, this average includes all samples. The issue is that the farther up the leading edge the time lookup goes, the more the deviation is between the true and wrong reference pulse.

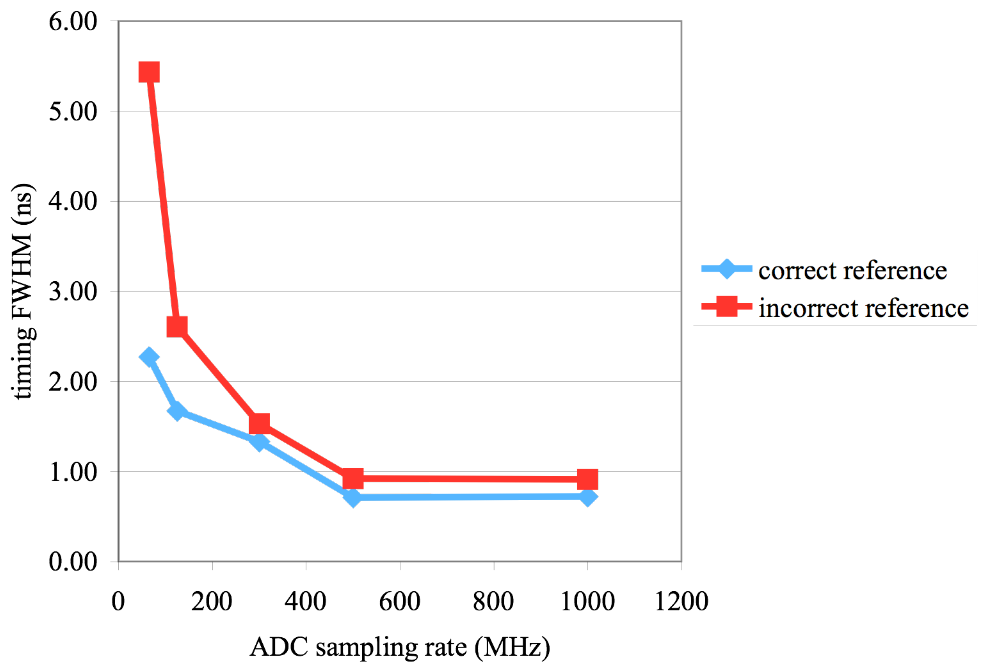


Figure 5.2. Timing resolution of our data set using the wrong reference pulse as compared to using the correct hand-tuned reference pulse.

5.2: Reference Pulse Discovery

Since it is difficult to manually obtain a good reference pulse with an oscilloscope, and using a single reference pulse for the entire scanner is unfavorable, an alternative should be created. The approach developed in this chapter is to use the FPGA to form a reference pulse from sampled pulses. To do this, the FPGA must be able to build an accurate reference pulse defined at a higher resolution (about every 60ps in this work) from pulses that are sampled at the ADC rate (~15ns to 1ns). This method has multiple advantages. The first is the acquisition chain in Figure 5.1 no longer needs to be modeled, as any effect this 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 its 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 impractical. Furthermore, with offline techniques, 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; our automation of the FPGA reference pulse discovery allows for easy investigation of improved filtering, new detectors, or any other changes to the acquisition chain.

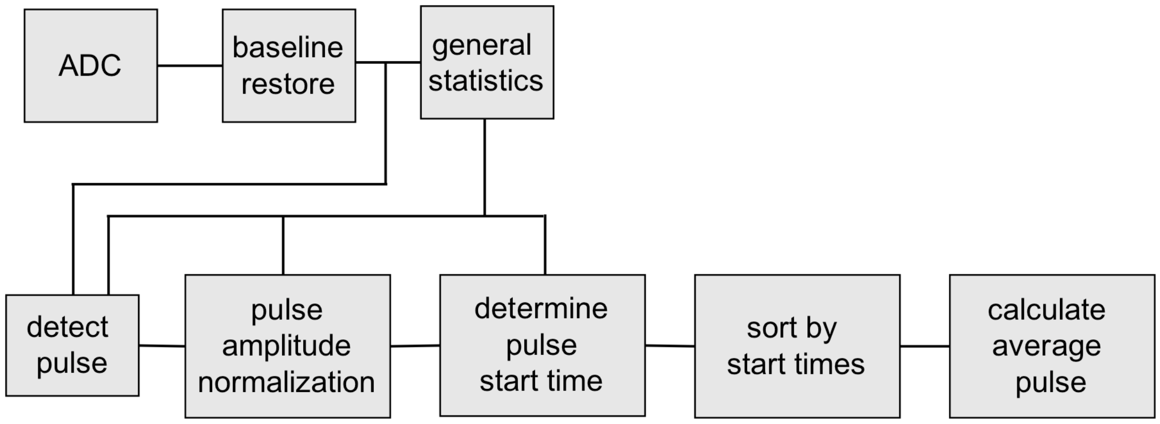


Figure 5.3. Block diagram of the overall reference pulse discovery algorithm.

The overall algorithm is shown in Figure 5.3. As samples arrive from the ADC, the first step is to remove the baseline as was described in chapter 4. With the baseline removed, the following general statistics are determined in the order as shown in the "general statistics" block:

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

After these statistics are calculated, pulses can be detected and processed. Pulses are detected by looking for eight consecutive samples over the event trigger threshold voltage (eight samples is for a 65MHz sampling rate and should be increased for higher sampling rates to guard against noise spikes). 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.

5.2.1: 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, as well as any other processing algorithm, should be automatically calculated. The parameters necessary for these algorithms are the voltage threshold used to differentiate pulses from noise, the average pulse length, and the average pulse energy.

It turns out that a good threshold can be calculated from the baseline window. Specifically, the threshold is set equal to the half of the window in the direction of photodector pulses. 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. Recall from chapter 4 that the baseline moves one lsb when the number of samples inside or outside of the window reaches a predetermined limit that is based on pulse length. At this point, the real pulse length is not known, so an estimate will have to be provided by the user. Waiting for the baseline window to expand and settle assures that the baseline window will settle before the rest of the algorithm starts. After the baseline window has settled, 1024 successive baseline window values (a new baseline value is calculated each clock cycle) 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) is counted until the voltage returns back below the threshold. This is done for 1024 pulses, and the average pulse length is calculated. One possible pitfall to this method is a large baseline shift during the time we are collecting the 1024 samples. If this occurs, the pulse length counter will keep counting until the baseline window has expanded to recapture the baseline. Depending on how long the baseline is out of the window, the counter may roll over, so a random value may be in the data set. A safeguard for this issue is to limit the number of bits in the counter to be just enough to count the largest possible pulse length. This will prevent a large count from a baseline shift having a large affect on the average pulse length. Additionally, we have also chosen to only count a pulse every 65536 samples so any temporary abnormal activity in the baseline will only corrupt one event out of the 1024.

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 controls 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.

Assuming a 65MHz FPGA clock, it will take slightly more than two seconds to calculate the threshold, average pulse length, and average pulse energy. The threshold is calculated after the baseline window has had enough time to move half of the ADC resolution (2048 for a 12-bit ADC). The baseline window can move one ADC tick every 600ns (three times the pulse length), so it will take about 1.2ms for the threshold to be calculated. For the pulse energy and length, it takes 1024x65536 cycles to compute, so for a 65MHz cycle this is 1.03s for each value.

5.2.2: Reference Pulse Formation

Once the general 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. This is accomplished by detecting when eight consecutive samples are above the threshold, just like in the previous steps. In addition to using the threshold, a narrow energy qualification around the average pulse energy is used. That is, only pulses within a narrow range of energy are used. The energy window is based on the average energy calculated in the previous step. For the simulations in this chapter, the window was chosen to be the average pulse energy ±1/8 of the average pulse energy. Only processing pulses within this narrow energy window guards against pulses from Compton scatter or pileup events from being a part of the reference pulse. The downside of using a narrow energy window is that it may take longer to collect the 4096 pulses.

Once a pulse is detected and energy qualified, the amplitude must be normalized to a common amplitude. This is accomplished by normalizing the sum of the samples of the pulse, as was done in the timing algorithm.

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 previous ADC sample to starting just before the ADC sample. As illustrated in Figure 5.4, this will result in the first sample of the pulse having a range of voltages.

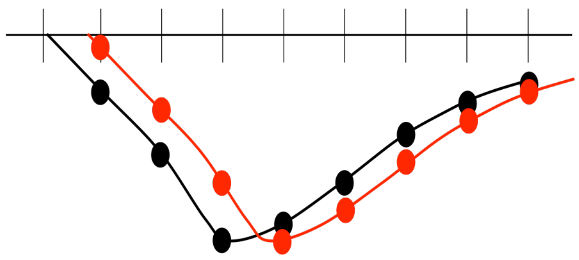


Figure 5.4. Illustration of the range of possible sampling of photodetector pulses.

To build the higher resolution reference pulse, each ADC sample interval is divided into many sub-intervals. The exact number depends on the ADC sampling rate and the desired precision of the reference pulse. For example, a sampling interval of 15.4ns (65MHz ADC) is broken up into 256 steps that are about 60ps each. An array with a length of 256 times the length of the 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 the algorithm is in the process of actually building this reference pulse, an alternative method must be used. For this step, we use a linear interpolation. For each pulse, the first two samples above the threshold are used to calculate a line, and the time of the zero crossing is interpreted using that line.

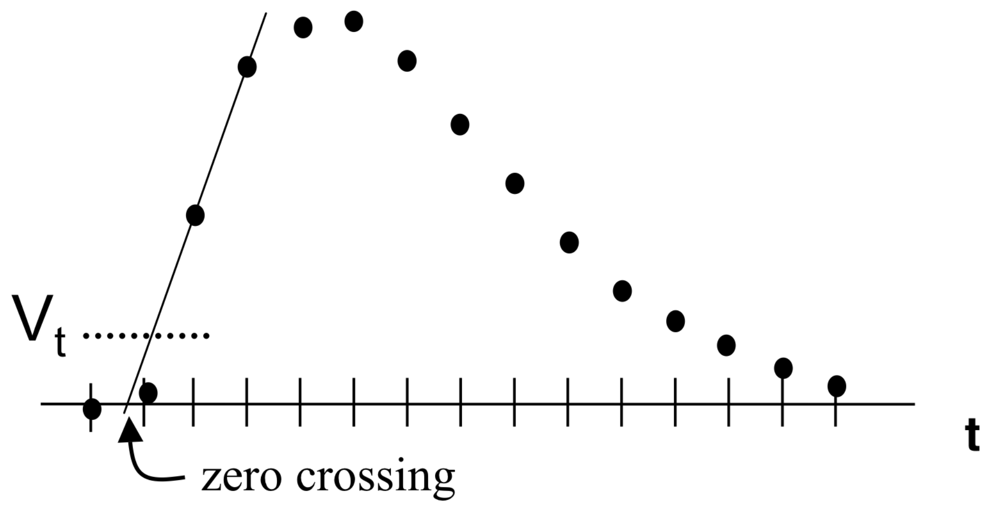


Figure 5.5. Linear interpolation of the first two points above the threshold (V_t) to determine the start time in the first iteration.

The fine-grain timestamp that results from the linear interpolation will indicate into which of the 60ps bins to place the first sample. The next sample of the pulse is placed in the same bin in the next ADC interval. In the example above, the second sample would be placed in the fine-grain timestamp plus 256. The third sample would be the fine-grain timestamp plus 512 and so on until all samples of the pulse are placed in the array. After many pulses are placed in the array in this manner, there will be a number of samples in each 60ps bin, as illustrated in Figure 5.6. This figure is a histogram of an ADC interval for a 65MHz and 300MHz sampling. The x-axis is the 60ps bins for one ADC interval, and the y-axis indicates the number of samples in each 60ps bin. The important feature of these histograms is how the samples are distributed across the ADC interval. It is imperative that there is essentially an even distribution of samples across the ADC sampling interval. In other words, a line that is fit to a plot should be fairly level across the interval. If the samples are not evenly distributed across the ADC interval (as illustrated in Figure 5.6a) the shape of the reference pulse may be incorrect, and timing resolution will degrade. For example, if there are few or no samples in the upper intervals (e.g. 220-255 for a 65MHz sampling rate) then the leading edge of the pulse will be too steep because the samples that are higher voltage and were supposed to be in these upper intervals are erroneously placed in the middle buckets. Likewise, if samples that are supposed to be placed in the upper intervals are placed in the lower intervals of the next ADC set, the slope of the leading edge of the pulse will be too shallow. Figure 5.6a shows that the distribution across the ADC interval is not even for a 65MHz sampling rate. There are more samples being placed in the lower part of the interval because of the linear interpolation timing on the first iteration. Recall that as the two samples used for linear interpolation get higher on the leading edge, the calculated start time is erroneously pushed back. A possible correction for this is to use a higher order interpolation such as a quadratic fit or use a timing method that doesn't rely on a reference pulse such as a DCFD. Figure 5.6b shows that a fairly even distribution was obtained at a 300MHz sampling rate (64 bins for a 300MHz sampling rate).

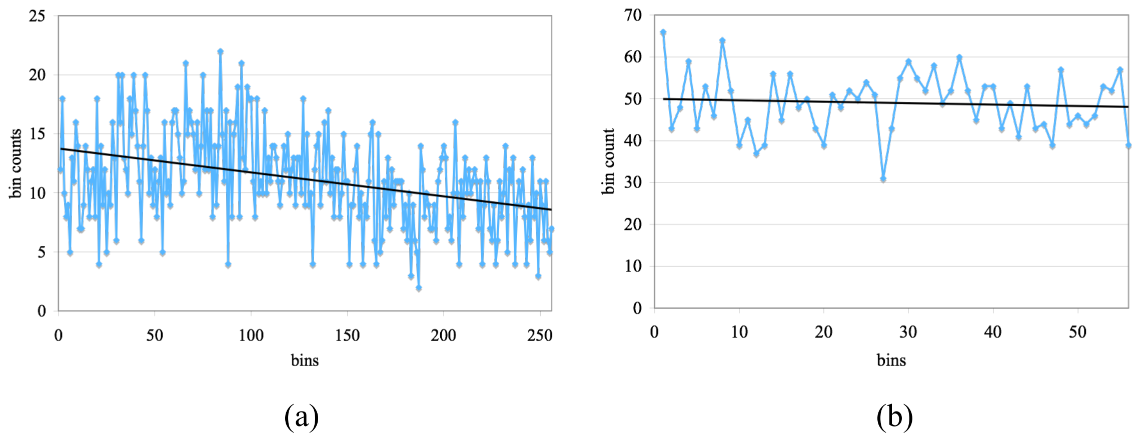
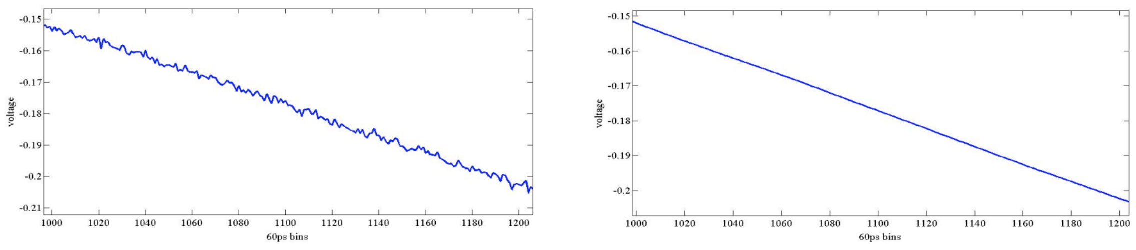


Figure 5.6. Histogram of the distribution of pulse samples over one ADC interval for the higher resolution pulse for a 65MHz (a) and a 300MHz (b) sampling rate. The solid lines represent a linear fit to the bin counts.

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. After this step, the pulse is still very rough (Figure 5.7a). To smooth the pulse, a moving average filter is used. Each bin is recalculated using the smoothing equation

$$v_i^{'} = \frac{1}{49}\sum_{i-24}^{i+24} v_i \, . \tag{1}$$

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 with the reciprocal. Figure 5.7 shows a close up the pulse before (a) and after (b) using the smoothing equation.



(a) (b)

Figure 5.7. MATLAB plot of 12ns of the leading edge of the derived reference pulse (a) before smoothing and (b) after smoothing for a 65MHz sampling.

One side effect of using the linear interpolation in Figure 5.5 to determine the start time of the pulse is that the initial portion of the reference pulse (first ADC interval) becomes a line. As illustrated in chapter 4, a linear interpolation is not a very accurate timing model, especially at lower sampling rates where samples are spaced farther apart (see Figure 4.6). To alleviate this, the process is iterated three more times. The only difference with the 2nd through 4th 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 a similar lookup technique as presented in chapter 4. Note that the baseline window and general statistics do not need to be recalculated. Each iteration uses the same set of sampled 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 (65 and 125MHz), the first section of the resulting reference pulse will be an exact match of the previous reference pulse. To eliminate this issue, two samples on the leading edge, as well as two samples just after the peak of the pulse, are used (Figure 5.8). The two samples on the tail of the pulse are used to assure that any errors in the slope of the leading edge will be corrected. Recall from chapter 4 that the tail of pulse provides poor timing resolution, so the last iteration only uses samples on the leading edge of the pulse.

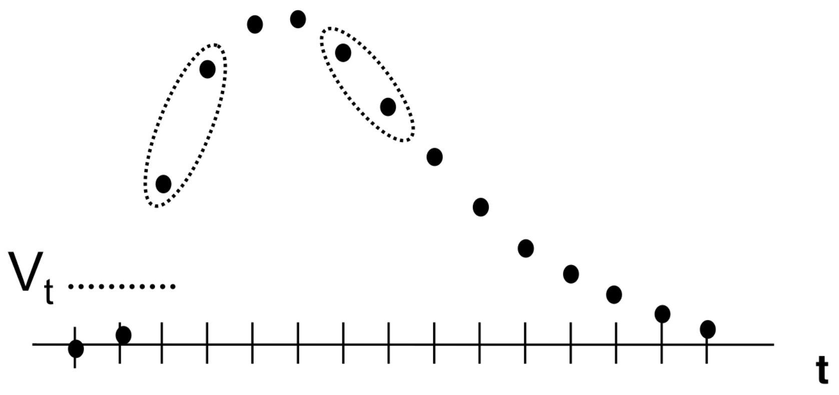


Figure 5.8. Illustration of the samples used in the timing step of the reference pulse formation.

To determine how many pulses are required, the algorithm was run with varying amounts of data pulses, and the resulting pulse was compared to the hand-tuned pulse from the same pixel. The comparison is the square of the error between the two pulses. That is, the two reference pulses were aligned in time and normalized for amplitude and the square of their voltage differences at each 60ps bin was calculated. The results are shown in Table 5.1. If the number of data pulses used in the tuning set is below 2048, the lower ADC sampling rates don't have enough data to fill out its bins with enough samples. For example, at a 65MHz sampling rate, if only 512 data pulses are collected for each iteration, and they are evenly distributed in each 60ps bin then there will only be two samples per bin on average. Essentially, Table 5.1 indicates that the only constraint is using less than 2048 pulses when using a 65MHz ADC sampling rate. The rest of the ADC and number of sample combination are statistically the same given the noise present in the pulses. Given these results, it appears that collecting 2048 pulses per iteration is sufficient. For our algorithm, we decided to use 4096 pulses because enough memory is available on the FPGA, and this provides a margin to assure that a decent number of samples will be present in each bin.

Figure 5.6 shows that even with a linear distribution, there can still be a large variance, so using more pulses gives a higher probability that none of the bins will be empty. This simplifies our algorithm so that it doesn't have to handle empty bins.

Table 5.1. Squared error between the algorithm defined reference pulse and the hand-tuned reference pulse for varying ADC sampling rates, as well as the number of data pulses used in the tuning set.

	number of data pulses used				
ADC rate (MHz)	512	1024	2048	4096	8192
65	7	4	0.006	0.003	0.004
125	0.07	0.01	0.008	0.009	0.009
300	0.02	0.01	0.01	0.01	0.01
500	0.06	0.05	0.04	0.05	0.05
1000	0.07	0.05	0.05	0.05	0.05

5.3 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 pixels from a Zecotech MAPDN photodetector using LFS-3; this is the same data set used in chapter 4. The pulses from a 511 keV (^{22}Na) source were collected with a 25GSPS oscilloscope and imported into MATLAB. Two classes of reference pulses were computed for each separate pixel. 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 (this was how the reference pulses used in chapter 4 were formed). The other reference pulse was calculated using the algorithm discussed above with pulses that were downsampled from 25GHz to 65MHz, 125MHz, 300MHz, 500MHz and 1GHz.

The first experiment is a subjective comparison of the two reference pulses – the hand-tuned one from the 25GHz oscilloscope data and the reference pulse derived from our algorithm with a 125MHz sampling rate. Figure 5.9 demonstrates that pulses can be derived from low-speed ADC data using our algorithm.

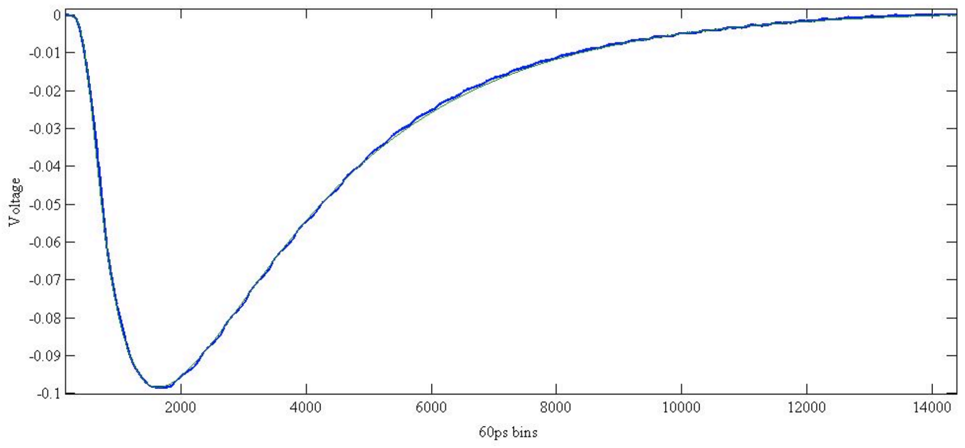


Figure 5.9. MATLAB plot of two reference pulses. One pulse is derived from our algorithm with a 300MHz ADC, while the other is derived from 25GHz oscilloscope data. The two pulses are so similar that it is difficult to distinguish them on this plot.

The second experiment compares the timing resolution using the reference pulses derived from the algorithm and the hand-tuned reference pulse. These results are compared to the results from chapter 4, which used a hand-tuned reference pulse. The same experiment that was done in chapter 4 for timing resolution was repeated with the algorithm defined reference pulse. The four data sets were paired up in all possible combinations, and coincidental pairs where simulated by aligning pulses in each data stream. Figure 5.10 shows the results of this experiment. This demonstrates that the algorithm is able to produce a reference pulse of the same quality at every ADC sampling rate, except at 65MHz where the timing resolution is 27% worse. This is mostly due to the inaccuracies of using the linear interpolation for the timing step on the first iteration. At 65MHz, the second sample can be close to the peak of the pulse, especially when the first sample is significantly above the threshold (i.e. the previous sample was just below threshold). The slope of such a line will be shallower as compared to a line that connects samples that are just above the threshold (see figure 5.6). In the former scenario, the line connecting the first two samples will indicate that the pulse started much earlier than it did. This will result in the samples being placed in a higher bin than the truth, resulting in a poorly shaped reference pulse after the first iteration. The subsequent iterations can recover slightly, but not enough to close the gap to the hand-tuned reference pulse.

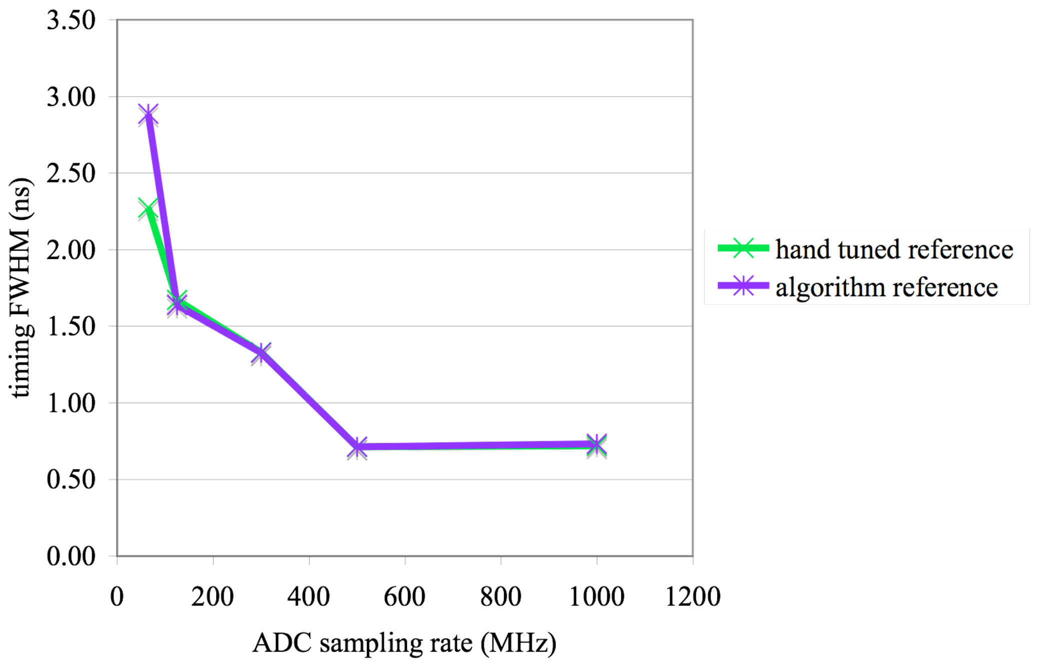


Figure 5.10. Result of coincidental timing for the hand-tuned reference pulse derived from 25GHz oscilloscope data and a reference pulse from our automated algorithm.

5.4: 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, the 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 and the multiple lookup tables. 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 a tuning circuit and does not need to be present on the FPGA during an acquisition scan, since only the parameters and leading edge of the reference pulse are needed during data acquisition. These values can be stored in off-chip memory before the FPGA is reconfigured with the acquisition circuit. 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.

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 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. So, the circuit has ample time to process each pulse at almost any reasonable clock rate.

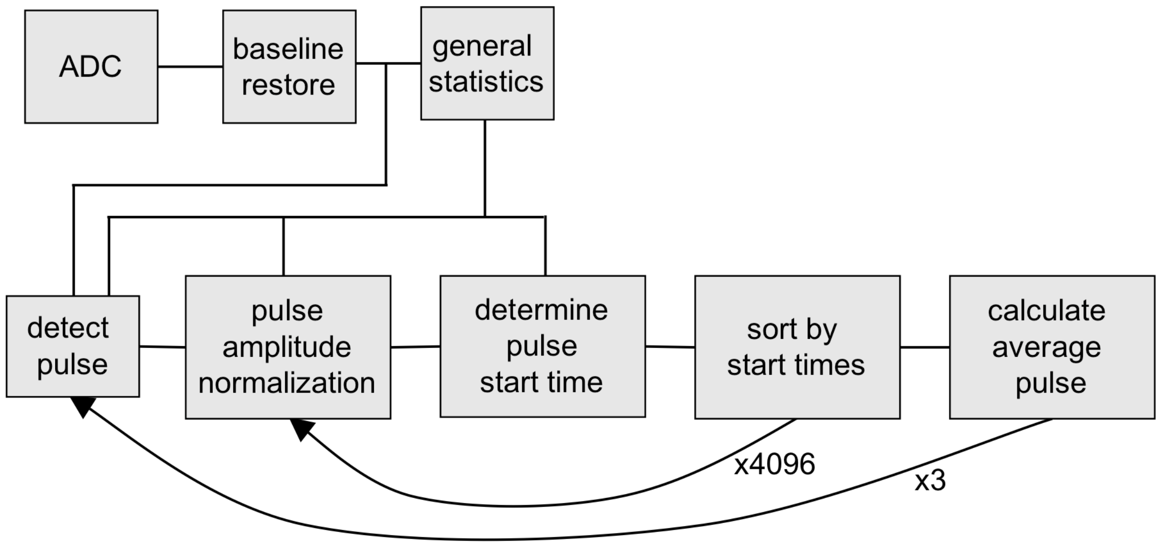


Figure 5.11. The final block diagram for our pulse parameter discovery showing that 4096 pulses are collected, and the whole process is iterated three times to refine the reference pulse.

Figure 5.11 shows the final algorithm. The calculation of the average pulse length and energy is straightforward. A counter keeps track of how many pulses have been collected. During pulse collection, the running sum of the pulse length and area are kept, and when all 1024 are collected the average is found by right shifting by ten (right shift by 10 in binary equals divide by 1024 in base ten). Since the threshold, average pulse length, and average pulse energy are calculated in succession, the same counter could be used for each.

As the pulses are detected using the threshold, the first step is to normalize them. The concept is the same as in the timing algorithm: normalize the area under the pulse. The complication in this algorithm is that the normalization lookup table cannot be precomputed as it is in the timing algorithm. This is because the reference pulse does not exist yet, so the value to normalize to is not known yet. The solution is to simply normalize all pulses to a predetermined value, and normalize the reference pulse after it is formed to that same value. This allows the reciprocal table to be precomputed, but a complication with this lookup table is that in theory, it is required to cover all possible pulse summation inputs, as it is not known a priori what the range of data pulses will be. 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 in 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.

Once a pulse is normalized for amplitude, the next step is to determine the start time of the pulse, which indicates in which of the 60ps bins to place the pulse. The complicated parts of this step is that two methods are used to determine that start time (linear interpolation for the first iteration and reference pulse inverse lookup for the subsequent iterations), and that the inverse lookup table has to be built after each iteration. To calculate the line that intersects the first two samples above the threshold, the slope and y-intercept need to be determined. The slope is calculated as

$$\frac{sample_{i+1} - sample_i}{fine\ grain\ bins \Big/ ADC\ interval} \tag{1}$$

where *sample*$_{i+1}$ and *sample*$_i$ are the voltages of the second and first sample above the threshold respectively, and *fine grain bins/ADC interval* is the number of times the ADC sampling period is divided into fine grain time steps. Recall that this number is predetermined and is a power of two, so the division is a simple right shift. Because we only need the fine-grain start time for each pulse to align them in the array, the y-axis (t = 0) is placed at the time of *sample*$_i$. This means that the y-intercept (where t = 0) is defined as the voltage of *sample$_i$*. So the start time can be calculated by

$$start = \frac{-b}{m} \tag{2}$$

where *b* is the y-intercept (*sample$_i$)* and *m* is the slope. Notice that the start time is a negative value, so *start* is subtracted from a large enough number that the earliest start time is a positive index into the 60ps array. The exact value of the offset is dependent on the level of the threshold. The higher the threshold is, the more negative *start* will be. This is a value that is currently user defined. Notice that it is better for the offset to be too large than too small, so that the index to the 60ps bin is always positive. A safeguard put into the FPGA is to check if the first index is negative. If it is, that sample is not placed in the array, but the rest of the pulse is placed at the appropriate indices.

A two dimensional array is used to store all of the 4096 sorted pulses. To compute the average value per 60ps bin, the bin's sum is read from memory, the sample is added to the value, and the sum is written back. A second memory of equal size is kept that tracks the number of samples in each bin. When all 4096 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. This division is performed with a lookup table of reciprocals. The reciprocal table is fairly small since there are relatively few samples per bin.

After the first iteration, the start time determination is done using the inverse lookup table, which means this table has to be built after the end of the previous iteration. The complication here is that the reference pulse is stored by times as it is being formed, but it must be converted to a voltage index as it is converted to the inverse lookup table. The conversion is accomplished by reading the average pulse array for each time *t* in succession. The resulting voltage for each *t* becomes the address for the inverse lookup table, and *t* is stored at that address. The complication is that the average pulse array is consecutive in time (i.e. has an entry for every possible 60ps bin), but there is no guarantee that the voltages will be consecutive. In regions of high slope (leading edge of pulse) the voltage between consecutive time values may be more than one least significant bit apart. When the voltages are converted to indexes of the inverse lookup table, they either need to be consecutive or the timing routine has to be able to handle missing voltages. If the latter was implemented, it may lead to a difficult lookup process, as it is not known how long it will take to find the closest valid entry. To create an inverse lookup table where all voltages have a valid time entry, any missing voltages need to be interpolated. If when the next time entry is read in the conversion process the voltage is greater than one least significant bit from the previous entry, then the times for the missing voltages need to be interpolated. Because the time is consecutive, it is know that the time of the missing voltages will be either the previous time index or the current. To determine which one to use, the number missing in voltage indicies is calculated. The idea is to place the time step essentially half way between the two non-consecutive voltage values (Figure 5.12). If there is an even number of missing voltage indices, then the lower half of the set is assigned the same time as the previous time index. Likewise the upper half of the missing voltage indices will be assigned the same time as the current time index. If there are an odd number of missing voltage indices, then it has been decided to give the lower half the extra time index. It is also possible to have duplicate voltages for consecutive time indices. In this case, the duplicated voltages are simply skipped. Note that if the voltage indices begin to decline, then the trailing edge of the pulse has been encountered. A second memory needs to be created to hold the trailing edge since the voltages (memory addresses) will have duplicates.

time	voltage
011011	001011
011100	001100
011101	001101
011110	010001
011111	010010
100000	010011

(a)

voltage	time
001011	011011
001100	011100
001101	011101
001110	
001111	
010000	
010001	011110
010010	011111
010011	100000

(b)

voltage	time
001011	011011
001100	011100
001101	011101
001110	011101
001111	011101
010000	011110
010001	011110
010010	011111
010011	100000

(c)

Figure 5.12. Illustration of the inverse lookup table conversion. After the average pulse is calculated and smoothed (a) the table is "flipped" (b) and the time for any missing voltage indices are filled in.

5.5: Conclusions and Future Work

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 parameters required for 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. Finally, we have 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 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.

The main difficulty in this algorithm is achieving the even distribution of the samples in the 60ps bins as indicated in Figure 5.6. If the distribution gets too skewed, the process cannot recover and the final pulse has the wrong shape. In fact, if the shape of the reference pulse gets too far off at any point, the problem can be exacerbated in the subsequent iterations. This imbalance can occur in any iteration, which is why it is important to use samples on both the leading edge and trailing edge of the pulse after the first iteration. The possibility that this problem can occur means that there should be a check after this tuning step. Fortunately, each references pulse does not have to be visually inspected after each tune. If one detector has particularly poor timing resolution or other parameters such as count rate or energy resolution are out of the expected range, then the reference pulse on that channel should be inspected. The inspection process would be to examine that a normalized data pulse has a similar shape as the discovered reference pulse. Additionally, the distribution of samples in the ADC interval as illustrated in Figure 5.6 gives a good indication of the quality of the reference pulse.

The initial implementation of this algorithm was implemented in Verilog as a dedicated circuit, but this algorithm is probably better suited for an implementation in the NIOS soft processor [19]. A soft processor is a microprocessor that is configured into the FPGA logic. It behaves and is programmed like a dedicated microprocessor, but it can also be tightly coupled with custom logic that is implemented in the rest of the FPGA. There is nothing in the algorithm that the processor can perform better; it is a matter of ease of implementation and code maintenance. Verilog is a powerful programming language, and is often necessary to get the best performance out of a circuit, but it can be difficult to write and complicated to change. The NIOS processor is programmed in standard C/C++, which is generally easier to understand, write and modify. The downside of using a soft processor in the FPGA over a custom circuit in the reconfigurable logic is the reduction in performance. The only circuits where performance is important is the baseline restore and pulse detection. These circuits need to keep up with the ADC rate and inspect every sample in real time, so these circuits would need to remain in the logic to assure the necessary throughput is achieved. The performance of the rest of the circuits is not a concern for this algorithm since it is a tuning algorithm, run very infrequently. Recall that there is a large delay between pulses that are processed by the design, so the processor will have many cycles to process each pulse. Figure 5.13 indicates the parts of the algorithm that can be implemented in software in the NIOS soft processor.

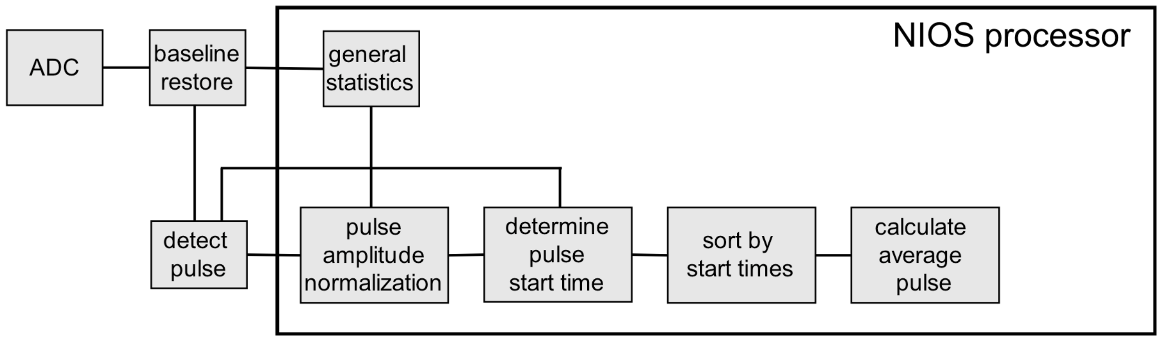


Figure 5.13. Block diagram of the portions of the algorithm that could be implemented in the NIOS soft processor.

Chapter 6 : Pulse Pile-up Correction

The previous chapters of this thesis largely ignored a complication that is present in real world scanners: pulse pile-up. Pulse pile-up occurs when two or more photodetector pulses occur within a short time period on the same channel. The result is one pulse being overlapped by a portion or all of a second pulse (Figure 6.1). For the timing algorithm discussed in chapter 4, this will present two issues. First, the area under the pulse can no longer be directly calculated by summing up all of the samples of the pulse, since some samples contain content from more than one pulse. This will make the area-to-amplitude lookup incorrect. Second, the samples on the leading edge of the second pulse will have a voltage component from the first pulse. This will inject errors into the voltage to time lookup that is performed in the timing step.

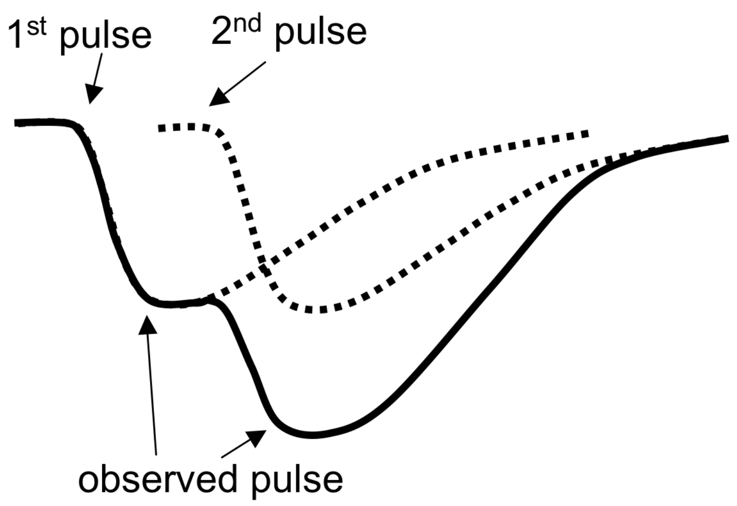


Figure 6.1. Illustration of pulse pile-up. The solid line indicates the observed pulse while the dotted line indicates the portions of the separate pulses missed because of the overlap.

Since radioactive decay is a random Poisson point process, pulse pile-up will occur in all scanners to some degree. The degree to which it will take place depends on a number of factors. The two main factors affecting pulse pile-up are pulse length and count rate.

The length of the photodetector pulse is determined by a combination of the intrinsic properties of the photodetector and scintillator, as well as the shaping characteristics of the channel that carries the pulse from the detector to the pulse processing hardware. The length of the pulse that is produced by the photodetector is part of the specification of the device. The pulse length of photomultiplier tubes range from about 10ns to 100ns, while silicon-based photodetectors tend to be about 100-200ns [20]. These times are in addition to the rise and decay times of the scintillator, which can add 4-600ns to the pulse length. After the pulse leaves the photodetector, it is often further stretched by electronics such as amplifiers and filters. As can be seen in Figure 2.4 the pulses from the photodetector contain a high frequency component that is generally removed with a low pass filter. In addition to removing the high frequency noise, the filters serve to slow down the rising edge of the pulse to assure an adequate number of ADC samples of the rising edge will be present. Also, the more you slow down a pulse, the better the energy resolution becomes; unfortunately this results in more pile-up, so a balance has to be obtained.

The largest factor in the amount of pile-up occurrences is the count rate on the channel being processed. This is dependent on the activity level in the patient as well as the architecture of the scanner. The activity is a function of how much radioactivity was present in the initial injection and the amount of time that has elapsed from the initial injection of the tracer. The architecture of the scanner dictates the effective area each channel covers. For a given activity level in a patient, there will be a certain photon flux per area. For example, a scanner that covers a square centimeter with ten discrete channels will have a lower count rate per channel than the scanner that has one channel per square centimeter. Of course the former scanner will have to process ten times more channels to cover the same total area as the latter. One such architectural feature that we are currently experimenting with is a common anode timing channel for an MAPD array [21]. A common anode is essentially a sum of all 64 pixels in the MAPD array; it produces a pulse if any one of the pixels detects an interaction. This means that the common anode has the same count rate as the entire array. Another detector feature that we are experimenting with that will affect channel count rate is a continuous scintillator crystal [22]. The continuous slab crystal replaces the array of discrete crystals. This presents a significant cost savings, but it creates some difficult problems for pulse processing, one of which is the increased chance of pile-up. This is because when a photon interacts with the scintillator, light is spread out over almost all of the pixels in the detector. Thus, if a second photon strikes the detector anywhere on the slab within the time it takes a first pulse to return to baseline, pile-up will occur in many of the pixels.

Pulse pile-up becomes an issue when a scanner is not set up to remove or correct it. The inability of a scanner to resolve two pulses that are close to one another is known as the scanner dead time. The scanner's dead time is defined as the minimum separation required between two pulses in order for the system to be able to distinguish them as two separate pulses. Dead time exists because of various factors in the pulse acquisition chain. The photodetectors can be a source of dead time if they require any reset after a photon interaction. For example, each pixel in a silicon photomultiplier is a bundle of straws that "fire" and emit a small current when hit with a photon from the scintillator. The number of straws that fire is directly correlated to the amount of light that hits the pixel. The straws have to reset after they fire before they will react to another photon, so if a second pulse occurs before all of the straws have reset, the pixel response may not be directly correlated to the amount of light from the second interaction.

The most common source of dead time is the pulse processing electronics. For example, a common timing pickoff technique utilizes a capacitor. At the beginning of each clock cycle a circuit starts charging the capacitor. When the output of the photodetector exceeds a set threshold, the circuit stops charging the capacitor. The amount of charge on the capacitor indicates where in the clock cycle the pulse arrived. Obviously, if another pulse arrives before the start of the next clock cycle, it cannot be given a time stamp because the capacitor has been turned off. Thus, this system has a maximum dead time of one clock period, and an average dead time of half of a clock period. In our system, the dead time is a result of the pulse processing algorithm and not the electronics. In fact, one of the advantages of an all-digital system with a free-running ADC is that in theory, it has no processing dead time. Of course this would require a system that has no limitation on the amount and duration of pile-up that it can correct. The timing algorithm of chapter 4 has a dead time of one pulse length, since the algorithm needs to sum up a clean pulse for the amplitude normalization step. It also requires the first couple of samples on the leading edge to be the true voltage, without having a component from the tail of a preceding pulse.

The goal of the work in this chapter is to reduce the dead time of our system in order to decrease the amount of lost data. In order to reduce the dead time of our system below the length of pulse, it must resolve pulse pile-up. Given our proposed timing algorithm, there are two types of pile-up; peak pile-up and tail pile-up (Figure 6.2). Peak pile-up occurs when two photons interact with a channel almost instantaneously. The result is that the pulses start so close to one another that the observed pulse appears to be one large pulse. Such pulses are almost impossible to separate, but should be detected and eliminated from the data set so they do not distort the energy spectrum. The second kind of pile-up is easier to recognize and mitigate. When tail pile-up occurs, there is a distinguished inflection point that occurs when the second pulse starts. In this work, we create an algorithm to detect and eliminate pulses involved in peak pile-up, as well as separate and process pulses from tail pile-up.

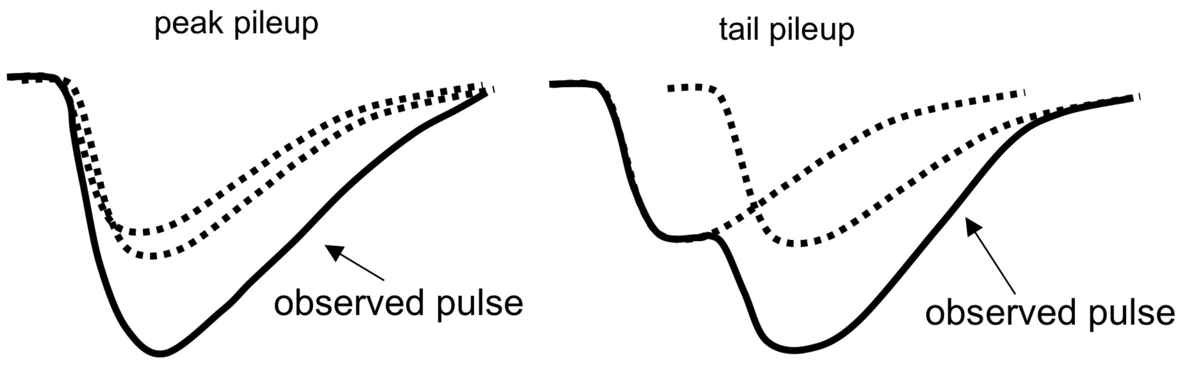


Figure 6.2. Illustration of the two kinds of pile-up.

6.1: Previous work

There have been previous analog and digital circuits implemented to recover from pulse pile-up. The first analog method uses a dynamic integration technique to retrieve the energy information from a pile-up event [7]. Dynamic integration uses an analog circuit to integrate a pulse until it detects a pile-up 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. A second similar but more straightforward method is the delay-line pulse-clipping (DCLP) method [8]. Instead of dynamically stopping the integration, DCLP electronically clips the tail off of all pulses at a predetermined time (often one decay constant) using an analog circuit. Since the signal is clipped, any pile-up that would occur after the cutoff is avoided. This reduces the likelihood of pulse pile-up occurring, but does not remedy any pile-up that occurs before the cutoff. This method also suffers from lower energy resolution, as only a portion of the overall information of the pulse is used, even in cases when pile-up correction is not needed. The high yield pile-up event recovery (HYPER) method corrects for multiple pulse pile-up 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 based on an exponential model of the pulse [23]. Essentially, based on the amplitude and the arrival time of the second pulse, the remaining energy of the first pulse is interpolated. This method requires analog components and two integrators that alternate integrating and discharging. This method has recently been converted into a digital implementation in an FPGA [9]. 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. Note that none of these previous efforts has studied the ability to timestamp the pulses involved in a pile-up event.

6.2: Algorithm

In contrast to the methods discussed above, we aim to develop a pulse pile-up correction algorithm on an all-digital platform that recovers the start time and energy of each individual pulse involved in tail pile-up. Our proposed method uses a combination of pulse shape discrimination and partial dynamic integration to detect and remove peak pile-up and to correct for tail pile-up. Pulse shape discrimination [24] is a technique that uses prior knowledge of what the pulses shape should be to determine when a pulse has been misshaped by a pile-up event. A simple form of pulse shape discrimination is done by fitting a reference pulse to the incoming pulse and determining that they match within a set of limits. As discussed in the timing chapter, fitting a reference pulse to the incoming photodetector pulse with a search is not feasible for a real-time implementation, so we have modified our timing technique to "fit" the reference pulse to the data pulse.

Figure 6.3 shows the general structure of the algorithm for pile-up 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 pile-up event present will be separated so that the second engine can process it for start time and energy. If no pile-up 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 pile-up event. The number of stages present in the algorithm is dependent on the amount of pile-up events expected. For example, if the probability of having a pile-up 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.

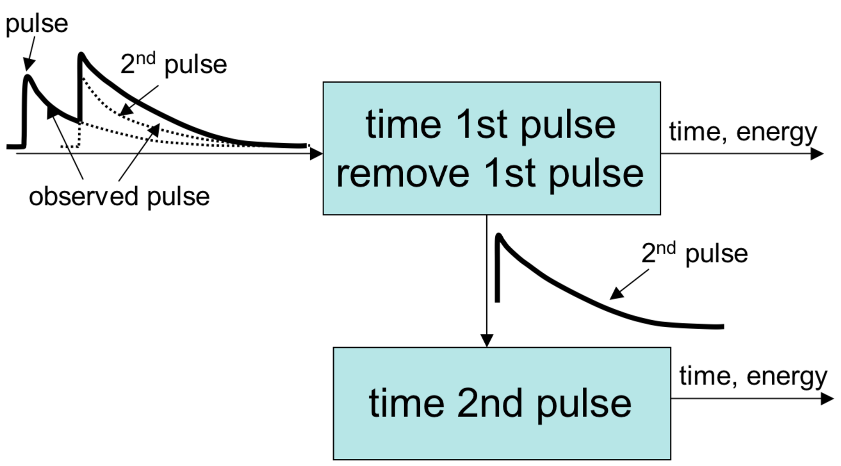


Figure 6.3. Block diagram of the overall pulse pile-up correction algorithm. The number of stages is dependent on the expected rate of pile-up.

To determine the start time of the pulse, the timing algorithm 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 pile-up. This means that we will try to mitigate any pile-up after this cutoff, and remove any pile-up 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 are 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. However, the time lookup does not have to be modified. To eliminate pulses involved in peak pile-up, 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 pile-up 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 pile-up 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. In this algorithm, it is assumed that pile-up 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 (about every 60ps in this work) than the sampling rate of the ADC. The data that is sent to the next processing engine is calculated using equation 7.1.

$$V[i] = V_{input}[j] - (V_{ref}[\left((n+i)\times\left({}^{t_s}\!/_{t_{ADC}}\right)\right) + \Delta t]) \times \left({}^{A_p}\!/_{A_r}\right) \qquad (7.1)$$

Here, $V_{input}[j]$ 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 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 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 pile-up 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 in equation 7.1. The cutoff point was experimentally determined with timing simulations that 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 pile-ups. 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 pile-up events we want to resolve. 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 pile-up train of length n follows a Poisson distribution, so even when a channel sees one million pulses per second, the likelihood of a train of length 4 or longer is about .3%.

The details of this algorithm will be developed with the experiments in the following sections. The experiments will determine where to place the cutoff for peak pile-up, as well as how to extract a pulse from the stream.

6.3: Tests and Results

6.3.1: Determining Peak Pile-up Cutoff

The first step in developing this algorithm is to determine where to set the cutoff between peak and tail pile-up. We started by investigating how much of the leading edge of the pulse is needed to accurately calculate the area-to-amplitude normalization and interpolate 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 percentage of the pulse that was summed was swept from 100% to about 5%. Figure 6.4 shows the results of this study. A sample pulse is included to provide a reference of what part of the pulse is being utilized. 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

6% 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 we would expect.

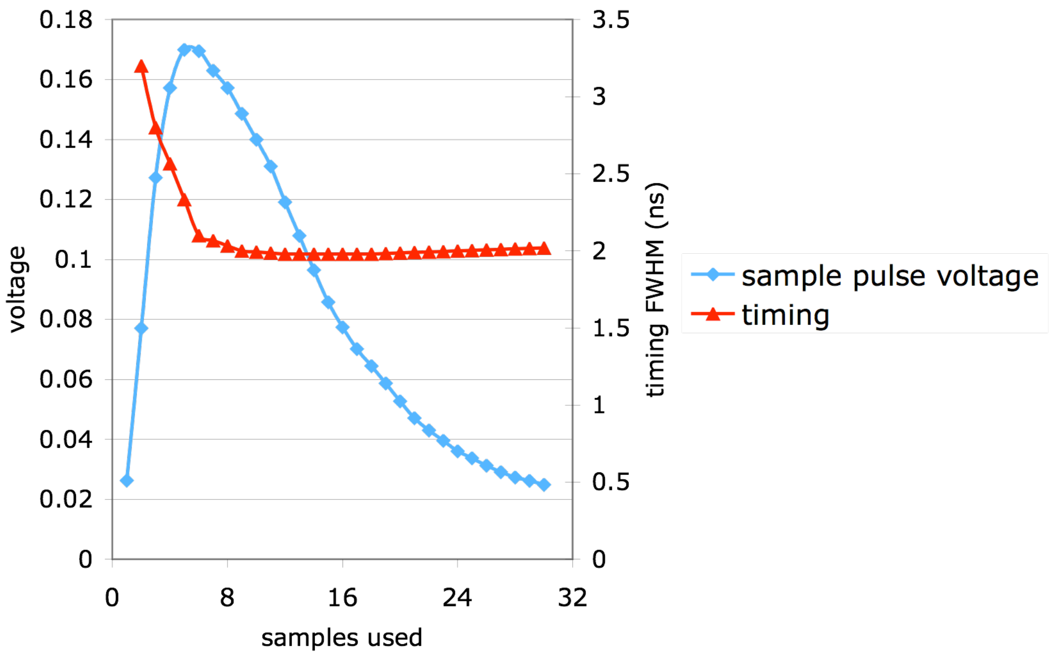


Figure 6.4. Graph of timing resolution for different amounts of the pulse used for amplitude normalization for a 65MHz sampled pulse.

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 standard deviation of the error for all 1000 pulse is plotted in Figure 6.5, along with an example pulse. The standard deviation demonstrates how much overall error is associated with interpolating the tail. Like the timing experiment above, 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.

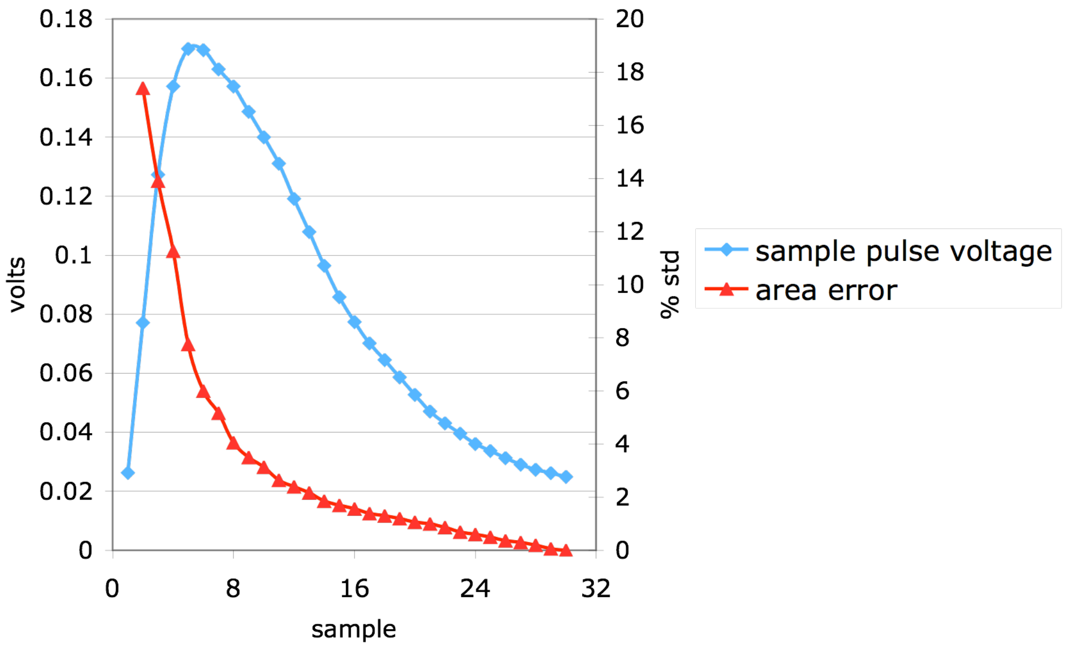


Figure 6.5. Graph of area error as compared to how much of the pulse is interpolated for a 65MHz sampled pulse.

The results in Figure 6.4 and Figure 6.5 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 Figure 6.5 will only be realized during a pile-up event because otherwise the entire pulse is summed. Given these results, the line that distinguished peak pile-up and tail pile-up is set to 20% of the pulse. That is, if pile-up occurs before 20% of the pulse, it should be detected by too much energy and discarded. If it occurs afterwards, it can be separated with our algorithm. Note that this is for this detector and may need to be reassessed for different cameras that may have different pulse shapes. In general, it seems that the cutoff has to be just beyond the peak of the pulse.

6.3.2: Area Correction

Even though the timing does improve as less and less of the pulse is summed for energy, there are some issues. One issue is the dependence of a pulse's calculated energy on the voltage of the first sample (Figure 6.6). Since the number of samples summed is always the same, a sampling where the first sample is well above the threshold (Figure 6.6b) will have a greater area than if the first sample was just above the trigger (Figure 6.6a) for the same pulse. Recall from the trigger discussion in the timing chapter that the range of possible voltages for the first sample of a pulse range from the trigger to some greater value based on the slope of the leading edge and the sampling interval. A greater range and more variance in calculate areas will be seen with a longer sampling period and a steeper leading edge.

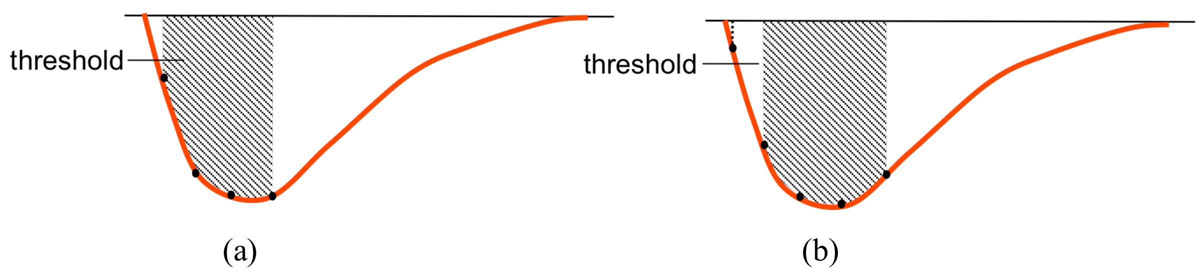
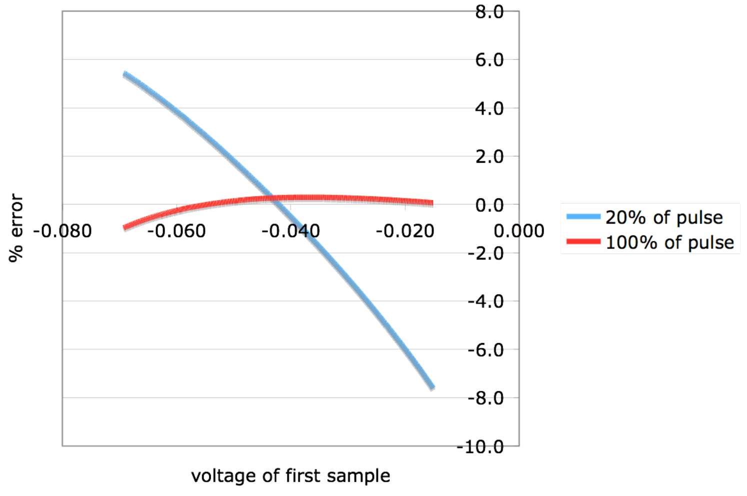
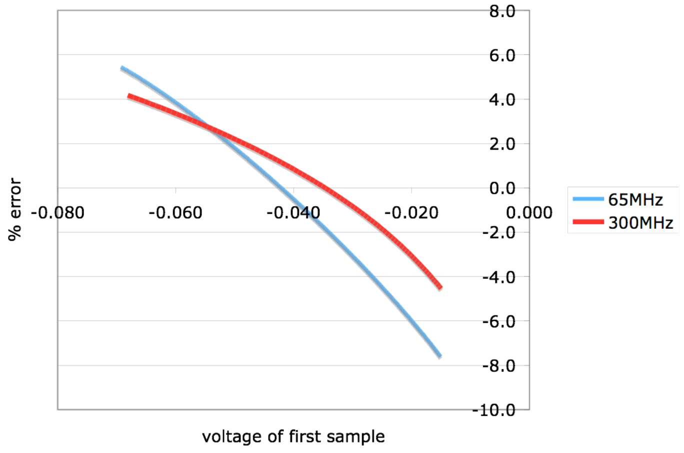


Figure 6.6. Illustration of how the calculated area of the same pulse can differ based on the voltage of the first sample.

Figure 6.7 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, Figure 6.7a 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. The range of possible first samples (determined by the ADC sampling rate) and the slope of the leading edge dictate the upper voltage. Figure 6.7a 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, 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. Figure 6.7b 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. 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 of the cutoff filter is higher. This is evident by the fact that the possible ranges of the first sample 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)

Figure 6.7. 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 expected error shown in Figure 6.7a is calculated using the reference pulse, and the correction is stored in a lookup table. The lookup table stores the difference between the average area of the reference pulse and the expected area obtained for the data pulse. Recall that the reference pulse is defined at about every 60ps so it has to be down sampled to match the ADC sampling period in order to accurately compare the reference pulse to the ADC data pulse. Based on the down sampling and the threshold, there is a theoretical range where the first sample can fall on the reference pulse. As illustrated in Figure 6.8, the lowest voltage is from a sample just above the threshold, and the largest voltage is when the previous sample is just below the threshold. The average area for the reference pulse is based on a sampling with the first sample is in the middle of this range.

Using the area correction value in the normalization step is an iterative process. The first sample that is used as the address to the look-up table has to be normalized first. So after the initial area of the first 20% of the pulse is calculated, the first sample of the pulse is normalized based on that area. The normalized voltage of the first sample is the address into the area correction lookup table, and the output is added to the initial area. Note that the normalized voltage of the first sample is not correct at this point, but it is close enough to use to look up the area correction value. The corrected area is used to normalize the first samples of the pulse again, which are used to determine the start time of the pulse. It is possible that this iterative process may need to be repeated more times for systems with larger area variances, but for this data set, one iteration sufficed.

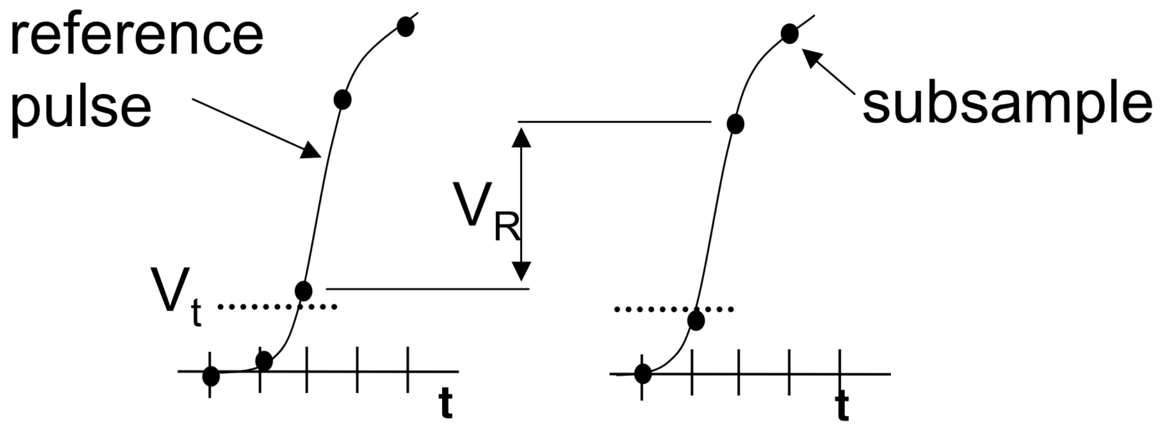


Figure 6.8. Illustration of the down sampling of the reference pulse. The range of first samples (V_R) is based on the ADC sampling rate and the threshold (V_t).

The results after correcting for the voltage of the first sample are shown in Figure 6.9 and Figure 6.10. 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 Figure 6.7a. The important difference in timing resolution and area error is when the summation limit approaches the peak of the pulse. For timing resolution in Figure 6.9, correcting the area improves the timing at 20% summation by 6% and the standard deviation of the area interpretation reduces by 53%.

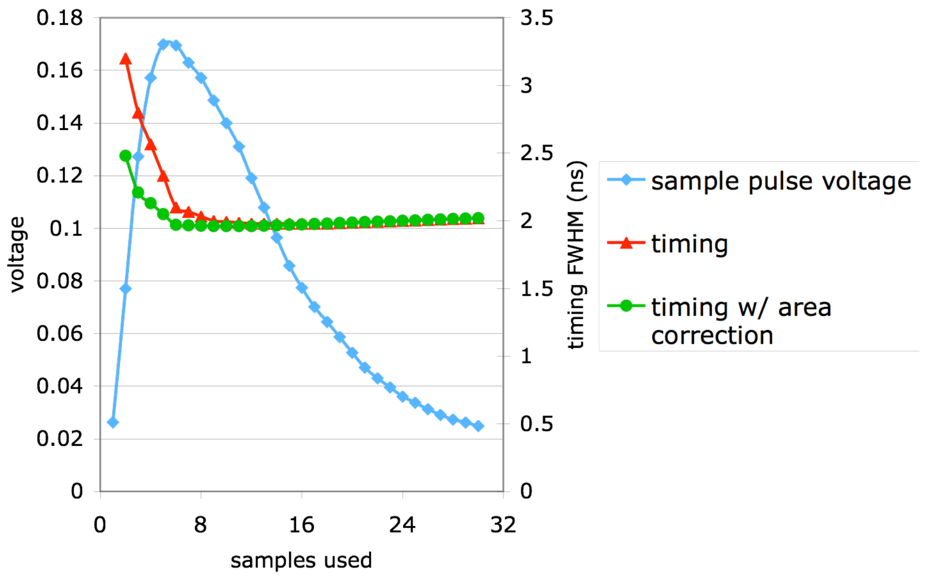


Figure 6.9. Graph of timing resolution vs. amount of pulse summed with and without the area corrected for a 65MHz sampled pulse.

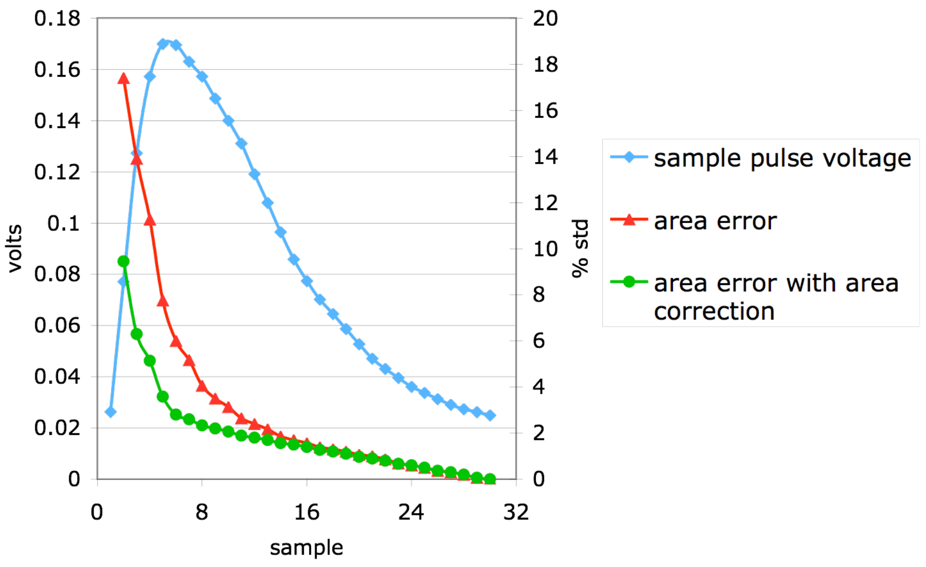


Figure 6.10. Graph of area error with and without area correction as compared to how much of the pulse is interpolated for a 65MHz sampled pulse.

6.3.3: 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 in timing chapter.
3) sum up the first 20% and the whole pulse.
4) check if first 20% summation is too large, indicating a pile-up event in the first 20% of the pulse.
    a. if the energy is too great stall until the stream returns to baseline and go to 1.
    b. otherwise continue.
5) normalize first sample 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
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 next pile-up correction engine
12) if next engine detects a pulse, this engine uses the energy based on 20% summed plus 80% interpolated
13) if next engine does not detect a pulse in the time it takes the first pulse to return to baseline, then this engine uses the 100% summation for the pulse energy and 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 pile-up 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. However, if 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.

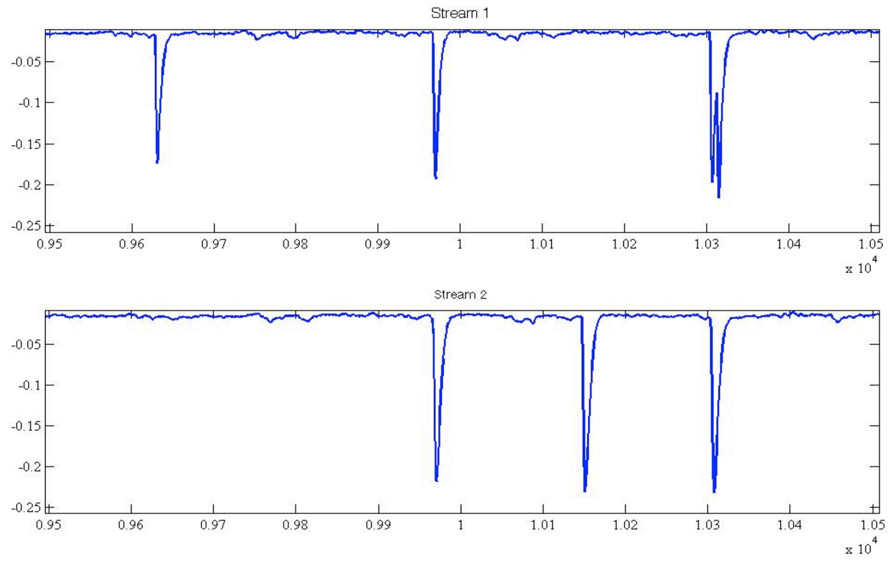
The array that makes up the data stream starts out filled with zeros, but as more pulses are added, the chance of two or more pulses overlapping increases. To generate a certain count rate, the length of the pulse stream is set such that after adding all of the coincidental pulses plus the random pulses, the number of pulses per unit time is correct. In this work we investigate 100 kilocount per second (kcps), 200kcps, 500kcps, and 1 Megacount per second (Mcps) (below 100kcps, pile-up 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 from the oscilloscope data are added to the zeros in the stream. A constant baseline is added to the pulses in the stream. This adds a baseline to the pulses without adding the baseline noise to the pulse, which would effectively double the noise. The final step is to filter the stream and sample it with desired ADC sampling rate. Figure 6.11 shows the pseudo-code for the stream generation routine.

```
Pulse stream generator
stream_length = ((1000/percent_coincidence) * pulse_length) / count rate
 for index = 1 to 1000
      start_time_1 = random(1:stream_length)
      coincidence? = random(1:100)
      if (coincidence? < percent coincidence * 100)
           start_time_2 = start_time_1
           pulse_1 = pulse_list_1(index)
           pulse_2 = pulse_list_2(index)
      else
           start_time_2 = random(1:stream_length)
           pulse_1 = pulse_list_1(random(1:1000))
           pulse_2 = pulse_list_2(random(1:1000))
      end

      baseline_1 = avg(pulse_1(1:300))
      baseline_2 = avg(pulse_2(1:300))
      pulse_stream_1(start_time1) = pulse_1-baseline_1
      pulse_stream_2(start_time2) = pulse_2-baseline_2
 end

 for index = 1 to stream_length
      if (pulse_stream_1(index) == 0)
           pulse_stream_1 = baseline_array(arrayIndex_1)
           arrayIndex_1 ++
      else
           pulse_stream_1(index) = pulse_stream_1(index) + avg_baseline
      end
      if (pulse_stream_2(index) == 0)
           pulse_stream_2 = baseline_array(arrayIndex_2)
           arrayIndex_2 ++
      else
           pulse_stream_2(index) = pulse_stream_2(index) + avg_baseline
      end
 end
 filter pulse
 sample pulse
```

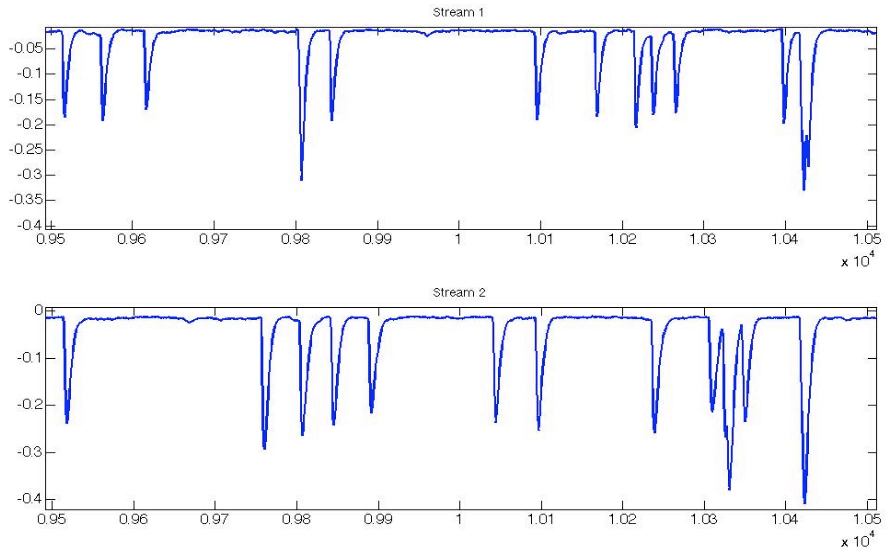
Figure 6.11. Pseudo-code for pulse pile-up stream generation.

The result of this routine is two streams that consist of randomly placed coincidental pairs with singles randomly placed in the stream. Figure 6.12 shows a small section of the resulting streams. Figure 6.12a shows two coincidental streams at 100kcsps while Figure

6.12b show the streams for 1Mcps. Notice that the first pulse in stream 1 in Figure 6.12a has no coincidental pair, nor does the second pulse in the pile-up event on the same stream.



(a)



(b)

Figure 6.12. MATLAB plot of the two pile-up streams from the pulse stream generator routine. (a) Streams with 100kcps. (b) Streams with 1000kcps.

These streams were created with a 50% coincidence rate for all possible pairings of the four pixels, 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 all possible pixel pairings. The results for how timing resolution is affected by the count rate are shown in Figure 6.13. 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 pile-up. The timing resolution from 100kcps to 1Mcps degrades by about 15% for a 65MHz-sampling rate and about 40% for the other sampling rates. Figure 6.13 also shows that the degradation is essentially linear over the sampling rates covered. This indicates that our algorithm degrades gracefully as pile-up rates increase.

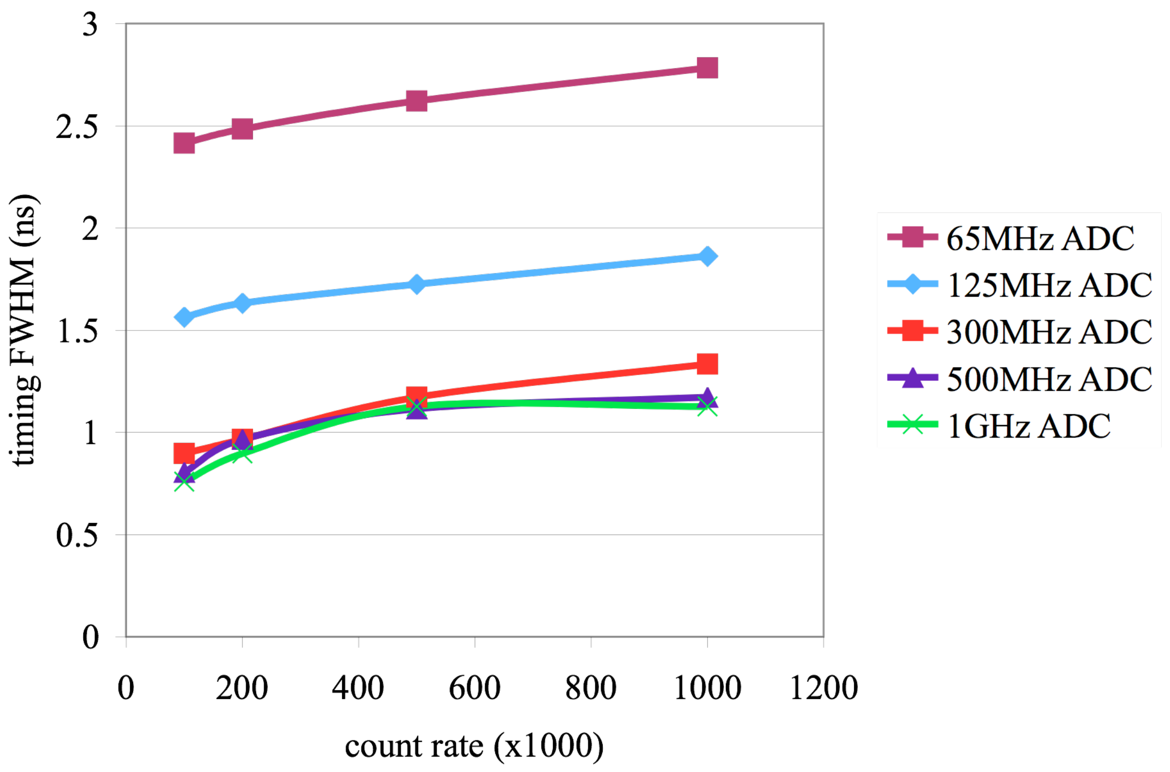


Figure 6.13. Timing resolution for different count rates at different ADC sampling rates.

To get an understanding of what is causing to the degradation, the timing resolution was calculated for each possible coincidence scenario. That is, coincidental pulses when neither pulse were involved in pile-up, when one of the two were in a pile-up event, and when both pulses were some part of a pile-up event. These results are presented in Figure 6.14 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 pile-up 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 pile-up events, and especially from the case when both pulses in coincidence are involved in pile-up. There is still some degradation in the timing resolution for pulses not involved in pile-up when count rates increase. The degradation is about 17% from 100kcps to 1Mcps. This is probably due problems calculating a good baseline when pulses are close together but not overlapped.

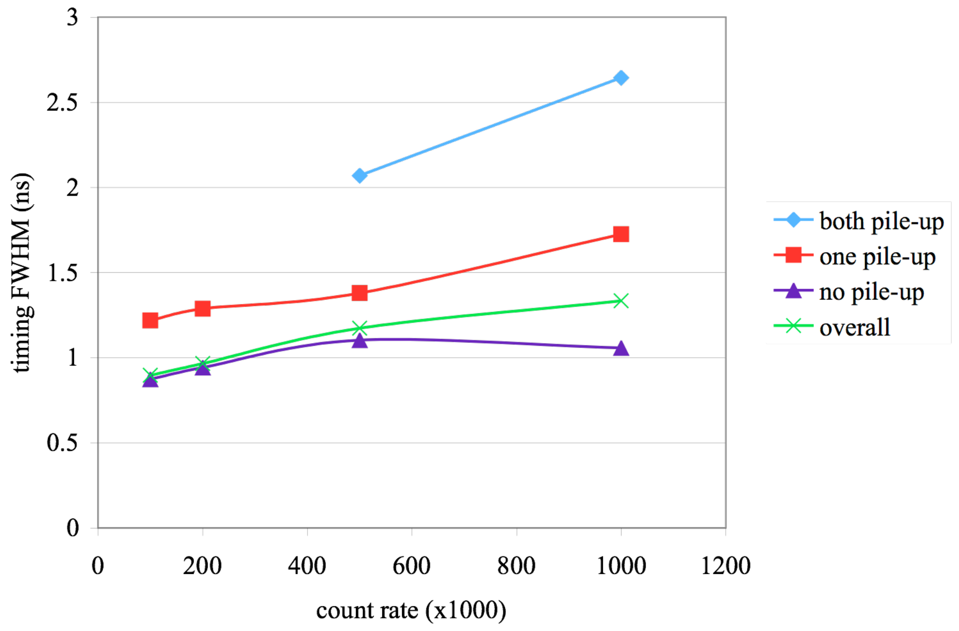


Figure 6.14. Plot of timing resolution for a 300MHz ADC sampling for different count rates. The timing resolution is shown for coincidental pulses where neither of the pulses is in a pile-up event, where one of the pulses is piled-up, and where both of the pulses had to pile-up. The overall timing resolution is the combination of all three.

In addition to timing resolution, energy resolution is an important factor of a PET pulse processing system. Energy resolution is a measure of the distribution of energy. A histogram of the recorded energies from a detector would have a shape as shown in Figure 6.15. The photo peak represents the energy deposited when 511keV photons deposit all of its energy in one interaction with the scintillator. This is referred to as photoelectric absorption. The remaining energy components arise from a number of different sources. A large source is from Compton scatters where only a portion of the photon's energy is absorbed in the scintillator. This results in a smaller amount of light production from the scintillator and a lower energy photon continuing at some incident angle to the original photon. The scattered photon can interact with the same or another scintillator, or it may not interact at all and escape from the scanner. Another source of non-photoelectric energies is from other photons present in the environment, as well as photons from the radioactivity naturally present in most scintillation materials.

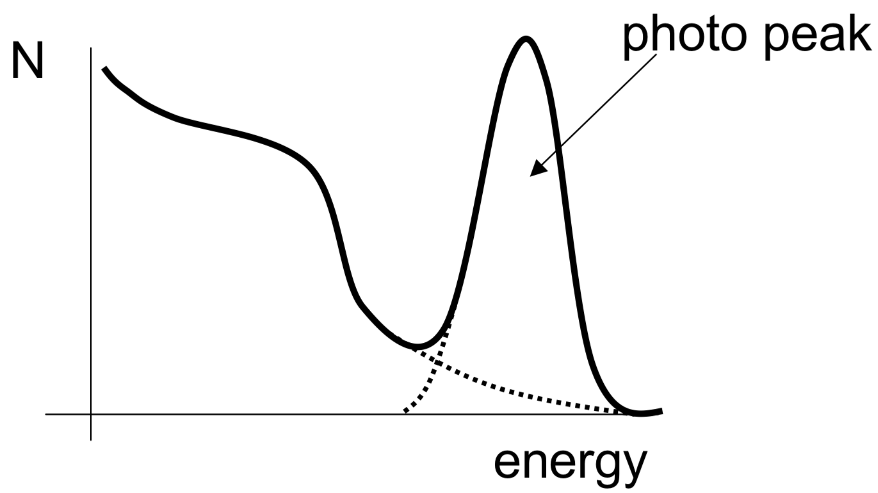


Figure 6.15. Illustration of an energy spectrum for a photodetector in PET for a source in air.

Notice that there is component of non-photoelectric energies in the photo peak because Compton scatters can deposit any energy up to 511keV, and the random photons can have energies in the photo peak. Obviously the random pulses should not be processed, but even the Compton scattered events generally cannot be used because it is not know if the interaction is the first or a subsequent interaction. To eliminate these unwanted pulses from the data stream, energy quantification is used. This process checks the energy of each pulse to see if it is outside the photo peak. The wider the photo peak, the more non-photoelectric pulses will be erroneously included in the data set. The width of the photo peak is dependent on a number of factors of the detection system. One source is the drift of the electronics recording pulses. Another source is the random noise present in the system. While these two sources are a product of the scanner design, the last source of energy distribution cannot be controlled: the statistical noise due to the discrete nature of detection system. For example, when a 511keV photon interacts with a scintillator, a discrete number of photons in the visible spectrum are emitted. The exact number will have a statistical distribution. Likewise, when those photons hit a PMT, the exact number of electrons that are produced will vary. All of this variation results in a Gaussian distribution of the photo peak. The width of the distribution is reported as the energy resolution. Energy resolution is calculated using equation 7.2.

$$R = \frac{FWHM}{E_0} \tag{7.2}$$

Here, FWHM is the full with of the photo peak at half of the peak's maximum value, and E_0 is the energy of the maximum of the photo peak. Dividing the FWHM by the peak energy means that the numeric representation of energy doesn't factor into the energy resolution. Energy resolution is a dimensionless value and is reported in percentages.

For the pile-up correction algorithm, the energy resolution will give an indication of whether the energy of the pulses involved in pile-up can be accurately estimated. To calculate the energy resolution, the energy of all pulses in the stream (except those with too much pile-up) was recorded and a histogram was generated similar to the one in Figure 6.15. 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 using 7.2. The energy resolution results are reported in

Figure 6.16. 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%.

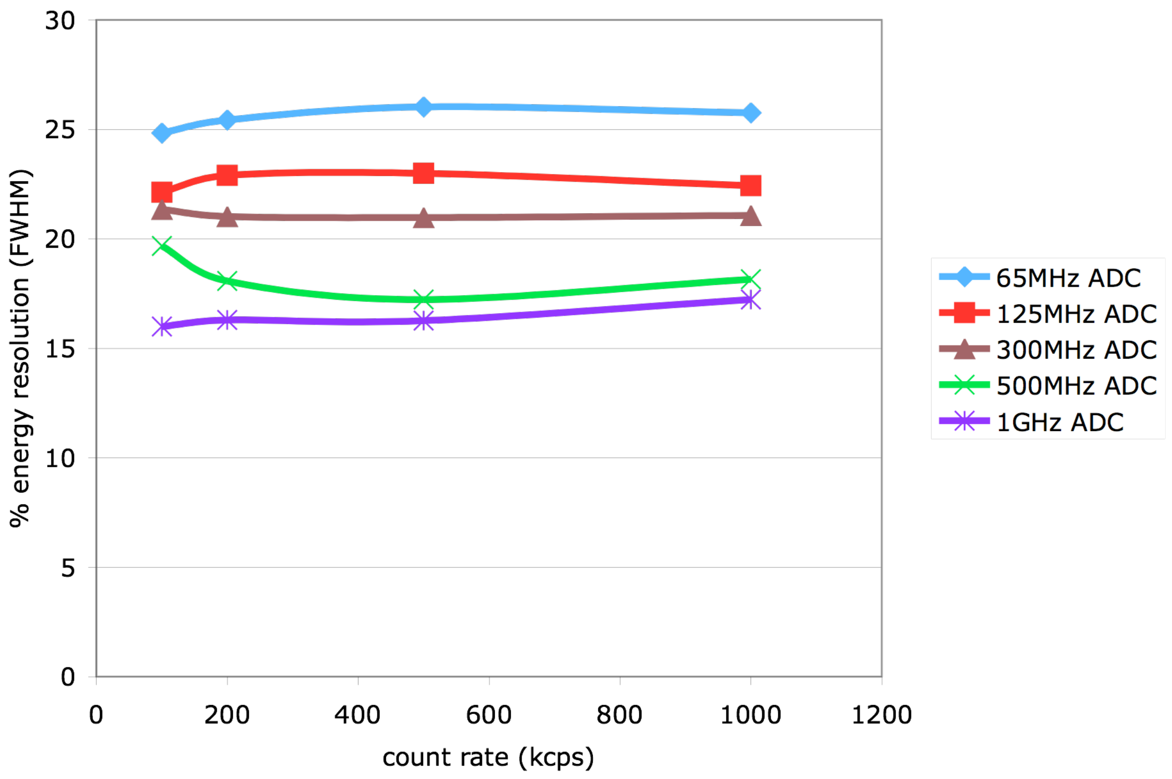


Figure 6.16. Graph of the energy resolution of our pulse pile-up correction as count rates increase for different ADC sampling rates

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 pile-up – pulse not involved in pile-up.
2) tail interpolate – first pulses in a pile-up event where the tail of the pulse had to be calculated with the reference pulse.
3) pile-up – pulses that are last in a pile-up event, where the previous pulse had to be subtracted out.

4) pile-up with tail interpolate – pulses that are in the middle of a pile-up event, where the previous pulse had to be subtracted out, and the tail had to interpolated from the reference pulse.

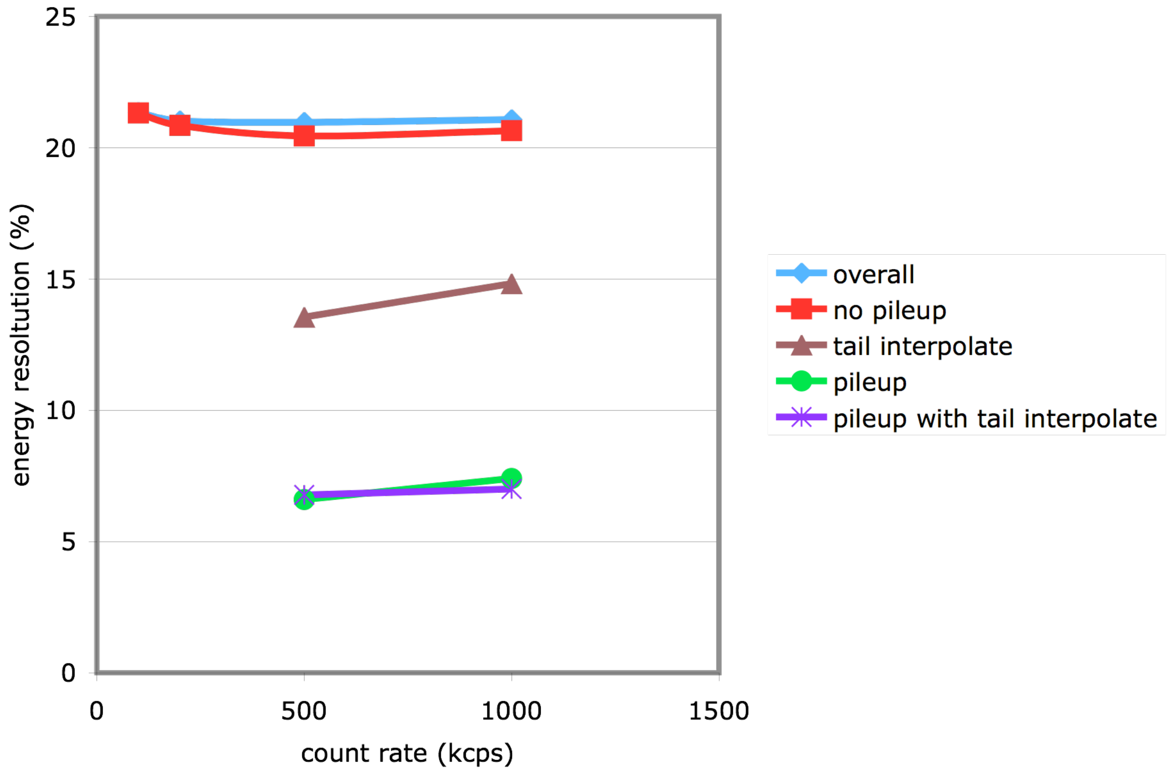


Figure 6.17. Energy resolution for a 300MHz ADC broken down to pulses in different places of a pile-up event.

Again, there are not enough pile-up events at 100kcps and 200kcps for any reliable results for the tail interpolate, pile-up or pile-up 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 pile-up are almost identical. Recall that when there is no pile-up, that the pulse is summed up to 100% for the energy calculation.

6.3.4: Peak Pile-up Filter

Determining whether a pile-up event has occurred before or after the 20% cutoff is calculated with the peak pile-up filter. Our first implementation relied only on the energy of the first 20% of the pulse. If the energy in the first 20% of the pulse was too large, it was deduced that pile-up occurred before the cutoff and that the two pulses should be discarded as peak pile-up. While this filter is very easy to implement, it is limited in its effectiveness. The problem with this solution is that it is only as good as the energy resolution of the system. In other words, if there is a large variance in the energy of the pulses, a significant portion of peak pile-up events will be passed through as tail pile-up. The reason for this inefficiency of the peak pile-up detection routine lies in the energy parameters of the pulses. Unfortunately, the range of energy of a pulse without pile-up varies by 70% 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 pile-up event (i.e. greater overlap) is required to trigger the filter. Therefore, this pile-up event will not be filtered as peak pile-up. This variance results in some peak pile-up being processed as tail pile-up when it occurs at 10% of the pulses length. A Monte Carlo simulation showed that only about half of the peak pile-up events were being correctly classified with this scheme.

Since the half of the pile-up that was correctly filtered occurred when the pile-up was in the first 10-15% of the pulse, a second part of the filter was added to that could detect peak pile-up between the 10% and the peak cutoff at 20%. This is accomplished by looking for a second rising edge before the peak pile-up cutoff. This scheme relies on the fact that the location of the peak is fairly consistent. So once the peak of a pulse is encountered (stored as a certain number of ADC samples) this filter looks for a number of samples that are greater than the peak value. The number of sample observed is dependent on the ADC sampling rate. For a 65MHz ADC, two samples are enough, but for a 1GHz sampling, eight samples need to be compared so that small noise spikes don't trigger the filter incorrectly. Notice that the filter doesn't look for samples that are all increasing (or decreasing for negative pulses), but looks for a number of samples that are greater than the peak. This slight difference means that if the second peak is encountered in this search, it will still trigger the filter. The effectiveness of this two-part filter is shown in Figure 6.18.

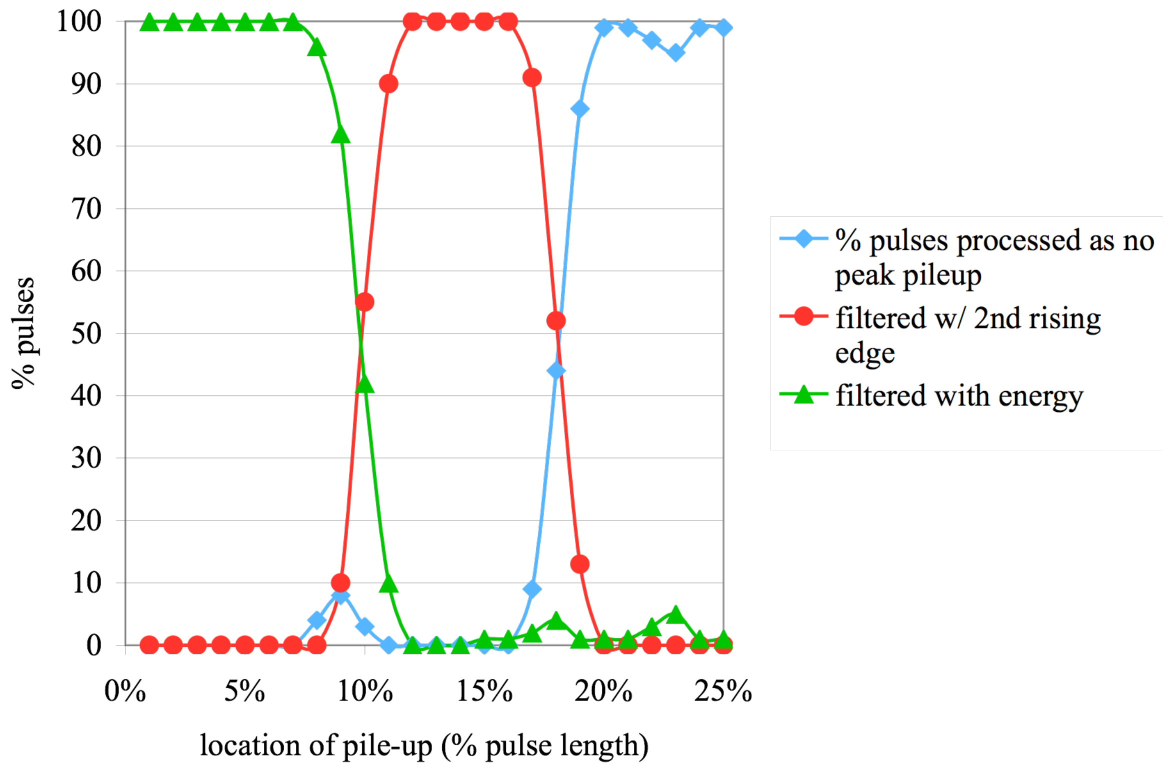


Figure 6.18. Plot of the peak filters ability to detect peak pile-up and percentage of peak pile-up filtered be each part of the filter.

Figure 6.18 shows the results from a simulation to test this peak pile-up filter. The simulation was generated where 200 different pulses from the same pixel were put into a stream so that 100 2-pulse pile-up events were generated. The degree of pile-up for all 100 events was the same for each test. The pile-up started at 1% of the first pulse (essentially two pulses directly overlapping) and was swept to 25%. In other words, 25 tests were run with all of the 100 pile-up events having the same amount of pile-up. For each test, the peak pile-up detection scheme was run and the number of pulses that made it through the filter (did not detect peak pile-up) was noted as well as how many pulses each part of the filter detected as peak pile-up. Ideally, the percentage of pulses processed as no peak pile-up would 0% until pile-up at 20% and then go immediately to 100% detected. Notice that the part that uses the energy in the first 20% of the pulse stops detecting peak pile-up when the second pulse is at about 13% of the first pulse. The reason for this is because some margin has to built in for the inaccuracy of calculating the energy of piled-up pulses. Fortunately, where the energy filters cuts off, the filter that looks for a second peak starts to detect peak pile-up. Before this point, the two peaks are close enough that the filter doesn't find a second rising edge. Notice that there is a slight bump in the number of pulses processed as tail pile-up around 10% because of the crossover between the two parts of the filter. Reducing the maximum energy value at the cost of possibly filtering out some "good" pulses can eliminate this bump.

6.3.5: System Characterization

An important character of a pile-up correction scheme is the resulting dead time of the algorithm. There are two types of dead time, paralyzable and nonparalyzable [20] (Figure 6.19). In a nonparalyzable system, the dead time is a fixed length and is not affected by a second pulse that arrives during the systems dead period. However, in a paralyzable system, a second pulse during a dead period extends the dead time. The different kinds of dead time can lead to a different number of pulses being detected. In Figure 6.19, where the dead time is one pulse length, five pulses are detected in a nonparalyzable system while only four are detected in a paralyzable system. The difference lies in the last set of pulses where three pulses pile up in a row. In the nonparalyzable system, the third pulse arrives after the dead time arising from the first pulse. However, in the paralyzable system, the second pulse extends the dead time beyond the start time of the third pulse. In this scenario, a nonparalyzable system is better, but is difficult to realize with a system based on a free-running ADC.

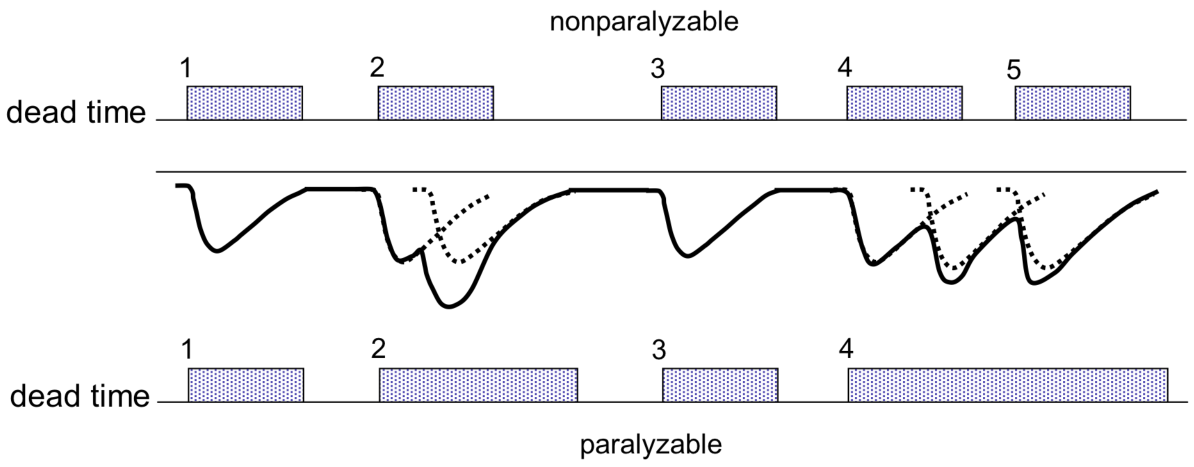


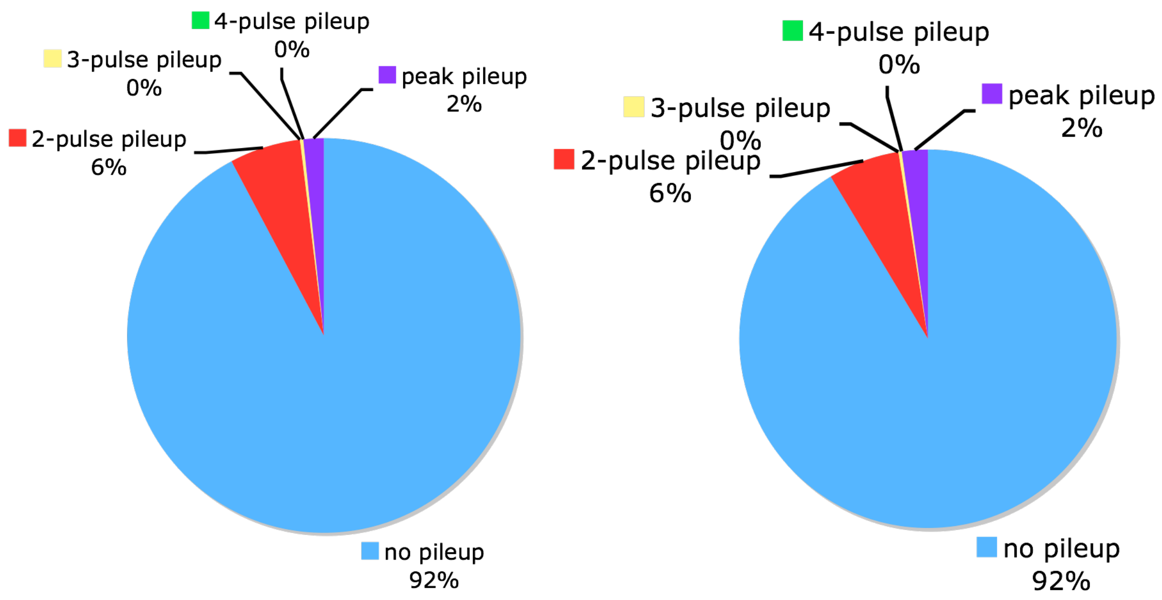
Figure 6.19. Illustration of paralyzable and nonparalyzable dead time [20].

Analyzing our pile-up correction algorithm indicates that it is a paralyzable system. Recall that if pile-up 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. This means that our system essentially has two dead times. The dead time when the system is live (not resolving peak pile-up) 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 pile-up 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.

6.3.5: Pulse Detection

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

Since our system has two dead times, the easiest way to calculate the expected pile-up rates is 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 was evaluated to determine how many pulse "groups" had no pile-up, and how many were a 2-pile-up, 3-pile-up or 4-pile-up event. The occurrence of too much pile-up was also determined. The results for the expected pile-up rates and the pile-up rates our algorithm calculated are shown for 500kcps in Figure 6.20 and 1Mcps in Figure 6.21. Note these charts don't show the percentages of individual pulses involved in different length pile-up events. To calculate this, the 2-pulse pile-up count has to be multiplied by 2, the 3-pulse pile-up event by 3 and so on. The number of pulses in the too much pile-up category is unknown however because the algorithm can't detect pulses once a peak pile-up event is detected.



(a) (b)

Figure 6.20. Chart of the expected (a) and observed pile-up rates for 500kcps with a 300MHz ADC.

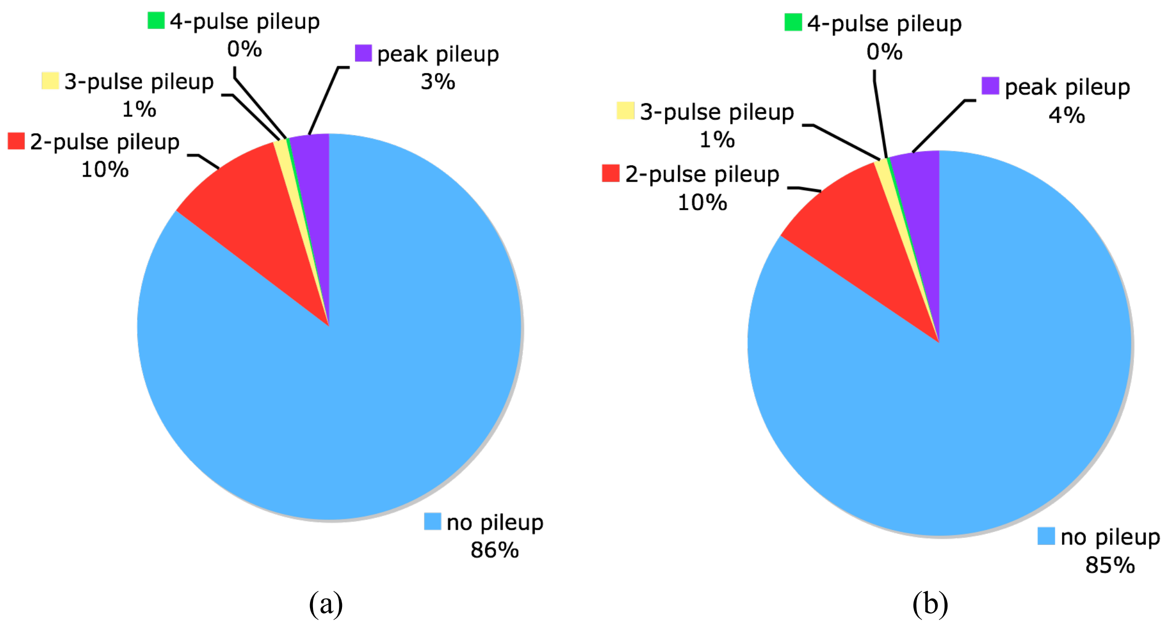


Figure 6.21. Chart of the expected (a) and observed pile-up rates for 1Mcps.

It appears that for both 500kcps and 1Mcps, the algorithm classified all of the pulses correctly to within 1%, except for the pulses involved in peak pile-up (i.e. pile-up in the first 20% of the pulse). Notice though, that this is for the total number of pulses detected. Table 6.1 shows the percent difference from detected to expected for each subcategory.

Table 6.1. Percent differences from detected pile-up events to expected pile-up events.

count rate	no pile-up	2-pulse pile-up	3-pulse pile-up	4-pulse pile-up	peak pile-up
500kcsp	-.2% (1702/1706)	5.3% (113/107)	20% (5/4)	0% (0/0)	23.8% (32/42)
1Mcps	-.9% (1442/1455)	-.6% (169/170)	-35.3% (17/23)	50% (8/4)	18.8% (56/69)

This algorithm does a good job at correctly finding pile-up (Table 6.1). The slight difference is mostly due to the way the expected pile-up rates were calculated versus the detected rates. The expected rates used the pulse length to determine if a pile-up occurred based on the start times of pulses. The pulse length is the same value for all pulses from a single pixel for implementation simplicity, even though the pulses do vary in length in practice. Additionally the stored pulse length is slightly less than it takes for the pulse to return to baseline for other issues. The algorithm on the other hand looks for a second pulse before the current pulse has decayed back to baseline. So even if the pulse length has expired, a pileup may be detected because the stream hadn't returned to baseline yet. This is why more pile-up events were detected, but fewer no pile-up events.

6.4: FPGA Implementation Feasibility

This algorithm was specifically designed for and efficient FPGA implementation, targeted to our second-generation hardware [25] that contains an Altera StratixIII S200. The goals for an efficient implementation are low logic and memory utilization and a real-time computation. Minimal logic utilization is required to accommodate the other pulse processing circuits present in the FPGA as well as the multiple channels supported. The real-time criteria is important because we don't want to add dead time to the system with the algorithms that are trying to reduce it.

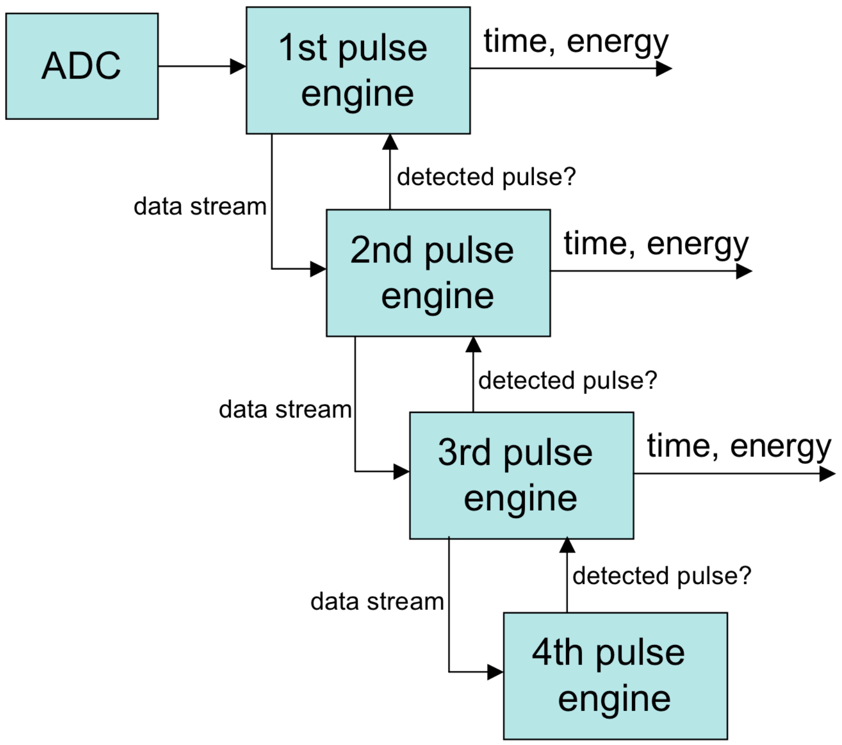


Figure 6.22. High level block diagram of a pulse pile-up implementation.

To create determine the FPGA implementation performance, graduate student Jim Pasko in our group developed an FPGA implementation [26]. His design was based on a modification of the timing circuit in chapter 4. In addition to the changes to the timing circuit required, the blocks are cascaded so each block is responsible for a given pulse of the pile-up event (Figure 6.22). The first change required is in the area-to-amplitude normalization step. The contents of this lookup table will need to be modified to reflect that the ratio is for the first 20% of the pulse and not the entire pulse. Another addition is the area correction factor that can be implemented in a lookup table. The final new piece needed is the block that interprets the tail and subtracts it from the stream. The block diagram for one pulse pile-up correction engine is shown in Figure 6.23. The number of engines is dependent on the count rate and the desired amount of correction. In addition to the correction engine, there is a termination engine that simply looks for the presence of a pulse so the prior engine knows whether to use the pulse energy based on 100% of the pulse or 20% plus the interpolated tail.

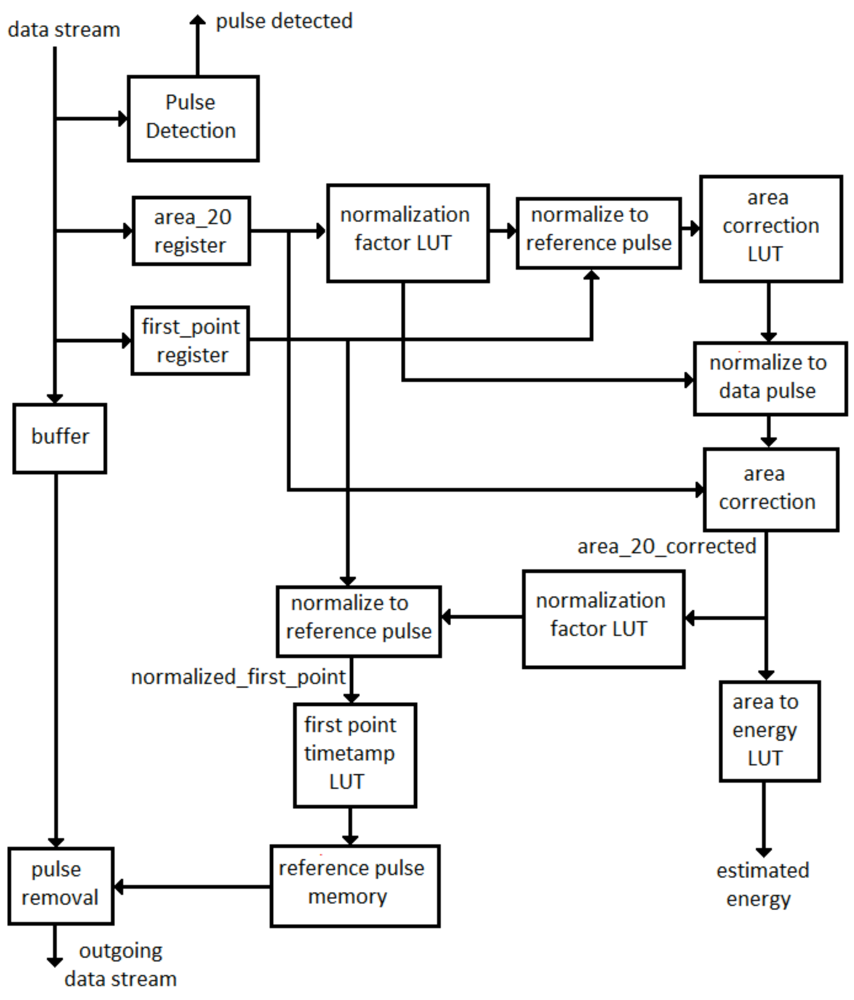


Figure 6.23. Block diagram of on pulse pile-up correction engine.

Most of these new logic blocks are pretty standard logic design. The challenges in this design all surrounded the latency through the engine. The first issue has to do with the control of the overall system. In our initial implementation, we had a control block that listened to each engine to determine when the first engine should be processing the stream, or when another engine is processing pile-up. This is important, because the first engine should trigger only the first pulse of a pile-up event, otherwise, if it triggers on any other pulse in the pile-up event, it will incorrectly process it. The problem with this control scheme is the latency through all of the engines. In other words, if it takes N cycles for a sample from the ADC to get through an engine, then the first engine will miss a multiple N samples from the ADC (depending on the number of engines). To correct this, the first engine simply looks for the ADC samples to return to baseline after it has processed a pulse. The downstream engines listen for the upstream engine to tell it that valid data is being passed down.

The other issue around latency is the coarse coincidence calculation. In our system [10], we utilize a coarse coincidence controller to lower the data rate to the host computer. The coarse coincidence controller is a separate FPGA that listens for events from all detectors and indicates when two events occurred in the field-of-view within the coarse coincidence window, which is three clock cycles. The problem occurs when two coincidental pulses are not in the same place of the pile-up stream. For example, if a single pulse was in coincidence with the second pulse signal of a pile-up event, the event signal for the single pulse would arrive at the coincidence controller N cycles before the other event due to the latency. This is corrected by delaying the pulse signal to the course coincidence controller from all of the engines except the last. The second-to-last controller has a delay of N while the next engine upstream has a delay of 2N and so on.

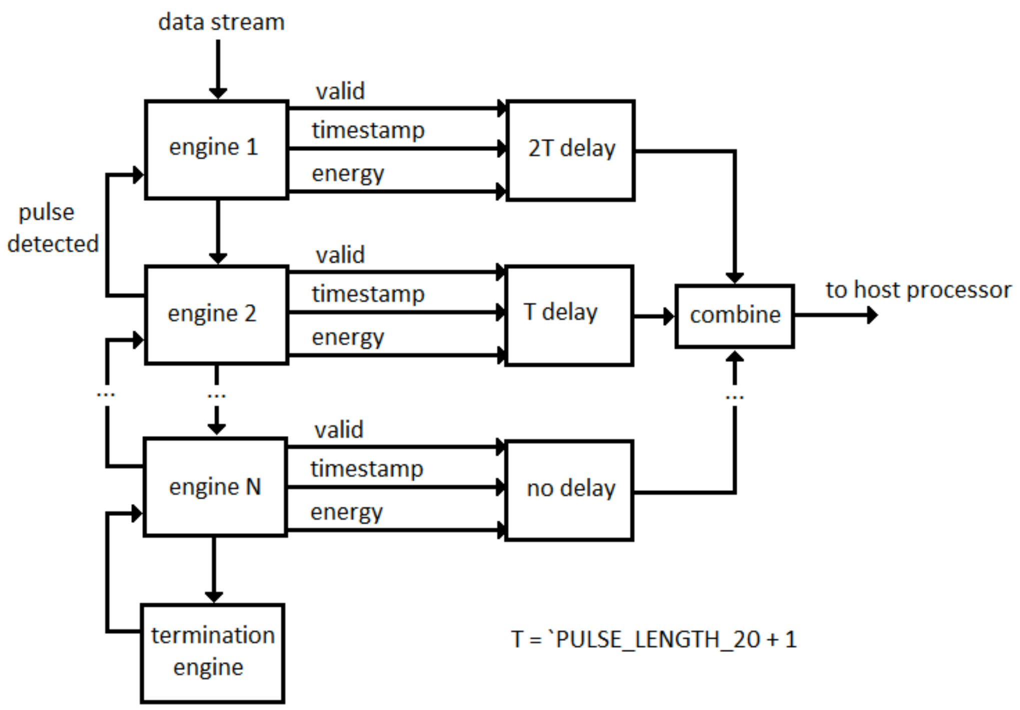
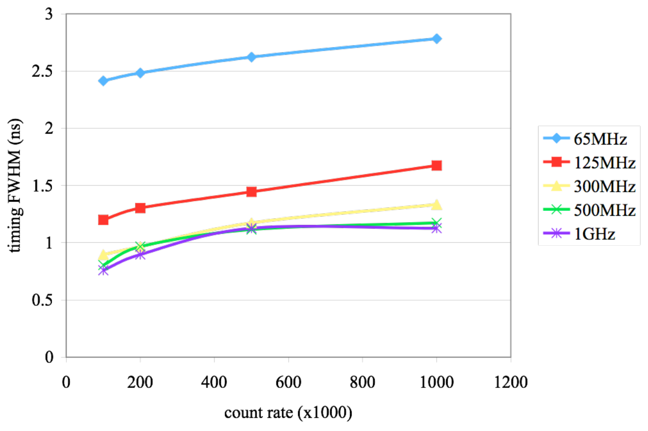


Figure 6.24. Delay scheme to correct engine latency for coarse coincidence timing.

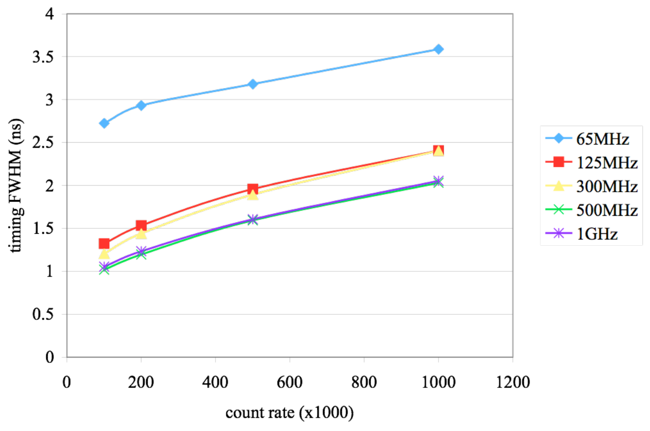
This algorithm was implemented in Verilog and compiled with Quartus for the StratixIII S200 FPGA used in our hardware. Each engine utilizes 217 LUTs (.1%), 337,284 memory bits (3.2%) and 6 DSP blocks (1%). The termination engine only uses 8 LUTs and 24 memory bits. Duplicating pile-up engines won't necessarily double the resources used. Some of the memories can be shared between two engines by utilizing the dual ports on the FPGA block memories. So for a system with two pulse pile-up correction engines, the memory usage is 552,592 bits. This represents a savings of 18% over simply duplicating the memory. There is also some possible memory savings by reducing the resolution of the reference pulse from 60ps to something less. This would require further study to determine the effect on timing and energy resolution.

6.5: No Pile-up Correction

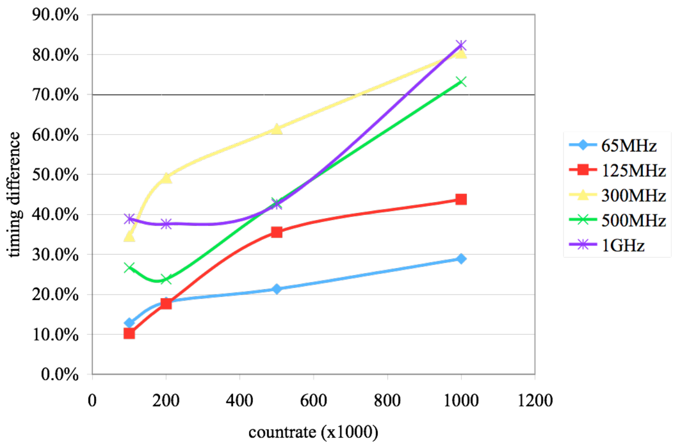
Given that the whole pile-up correction circuit will be at least four times larger than a single timing circuit, what would be the consequences of not doing pile-up correction in the presence of pile-up? In other words, how would the timing circuit presented in chapter 4 perform with the same data streams used in this chapter, which contain various amounts of pile-up. Figure 6.25 shows how the timing resolution degrades as pileup increases when neither pulse pile-up correction nor detection is used on a stream that has varying count rates. Figure 6.25a shows the timing resolution when the pile-up is detected and corrected. Figure 6.25b shows the timing resolution when pile-up is neither detected nor corrected. Figure 6.25c indicates the percent degradation between the two. At 100kcps, the timing resolution is 10-40% worse, and at 1Mcps, the degradation is up to 83%.



(a)



(b)



(c)

Figure 6.25. Timing resolution with pulse pile-up correction (a), and without pulse pile-up correction or detection (b), and the percent difference between the two.

Essentially, processing a stream that has pile-up without correcting or detecting it achieves timing resolution equivalent to using an ADC with half of the sampling rate. For example, the timing resolution of a 300MHz ADC without pile-up mitigation is the same as that of a 125MHz ADC with pile-up mitigation.

6.6: Conclusions and Future Work

This chapter considers a complication in PET detectors known as pulse pile-up, where two or more pulses from the photodetector overlap one another on the same channel. The freqeuncy that pile-up occurs is dependent on many factors, such as tracer activity levels and the scanner architecture. Since two architecture features present in our scanner will increase the rate of pile-up occurrence, this is an important addition to our pulse processing system. A common anode allows the use of a higher rate ADC for better timing resolution, and using a continuous slab crystal reduces scanner cost, but both of these designs have the side effect of greater pile-up rates. If left undetected, pile-up will corrupt the energy resolution of the system, as well as possibly lead to incorrect time stamps, both of which will lead to more random coincidence errors. If pile-up is detected, but uncorrected, it will result in lower detection efficiency. In our simulation up to 27% of the pulses were involved in pile-up in some manner. This loss of events will require longer scans to achieve the same number of counts for image reconstruction. Note that adding more activity to the patient will not necessarily work, as that would increase the pile-up rates. Previous efforts to correct pulse pile-up have relied in some part on an analog circuit. This requires extra circuits external to the FPGA, which draw more power, occupies board space, and requires extra I/O.

An alternative is to utilize the existing hardware and extend our timing algorithm to handle pulse pile-up. This is the approach that our algorithm takes. The pile-up mitigation algorithm presented in this chapter leverages some of the existing logic from the timing algorithm discussed in chapter 4.

This all-digital pulse pile-up correction algorithm attempts to detect and eliminate instances of peak pile-up, as well as separating tail pile-up to extract good timing and energy figures. We show how this can be done with a series of pile-up processing engines that extract timing and energy information from a pulse, remove it from the data stream, and send the resulting stream on to the next engine. We demonstrated that only 20% of the leading edge of the pulse is needed to extract good timing and energy information. Summing just 20% of the pulse created an error that was associated with the relationship between the ADC sampling and the start time of the pulse. This led to the introduction of a correction step that adjusts the area based on the voltage of the first sample. Also, because it is assumed that only the first 20% of the pulse is not corrupted by pile-up, a reference pulse is used to interpolate the remaining 80% of the pulse for an energy calculation, and to remove the pulse from the data stream. This can be done using the same reference pulse required for timing.

While being able to detect and process pulses in pile-up events will lead to higher detection efficiency, the processing has to be accurate enough to not degrade the overall system. If the timing or energy resolution is greatly degraded then too many coincidence errors will occur. The results from a simulation with real photodetector data indicates only a slight degradation in timing resolution and very little impact on energy resolution when almost 1/3 of the pulses are involved in a pile-up event. The timing resolution is about 40% worse when the count rate increases from 100kcps to 1Mcps.

The FPGA implementation [26] shows that a real-time solution can be achieved with relatively few FPGA resources. This is important for inclusion in a system with many other demands on the FPGA resources.

Chapter 7 : MiCEs Scanner Hardware and Overall Processing Architecture

While the algorithms in the three previous chapters have been designed to work on a generic PET scanner that has an FPGA with free-running ADCs as the processing core, there is a specific scanner hardware that was the intended implementation target [25,27]. The specifics of the scanner hardware place limitations on the algorithms. For example, the scanners lack of an analog trigger means that we can't develop an algorithm that depends on an external trigger. Additionally, the scanner hardware will dictate the design of the signal processing architecture. For example, the number of channels each FPGA processes and the count rate of these channels will indicate how many timing circuits will be required in each FPGA. This chapter will present the target hardware for this thesis to get an understanding of how the hardware limited the algorithms and vice versa. Additionally, in consideration of the scanner hardware, the overall organization of the algorithms presented in this thesis will be discussed. This will demonstrate how these individual pieces will operate together to create a functional pulse-processing platform for our PET scanner.

7.1: MiCEs Scanner Hardware

We are in the process of developing a set of data-acquisition electronics that are based on a free-running ADC and a modern FPGA for a small animal PET scanner. The hardware has been designed to leverage the strengths of the FPGA to create an all-digital solution and to improve the signal processing capabilities. The general architecture of the scanner hardware is illustrated in Figure 7.1. In addition to the FPGA and ADCs, there are other peripherals that are required to support the algorithms that will reside in the FPGA.

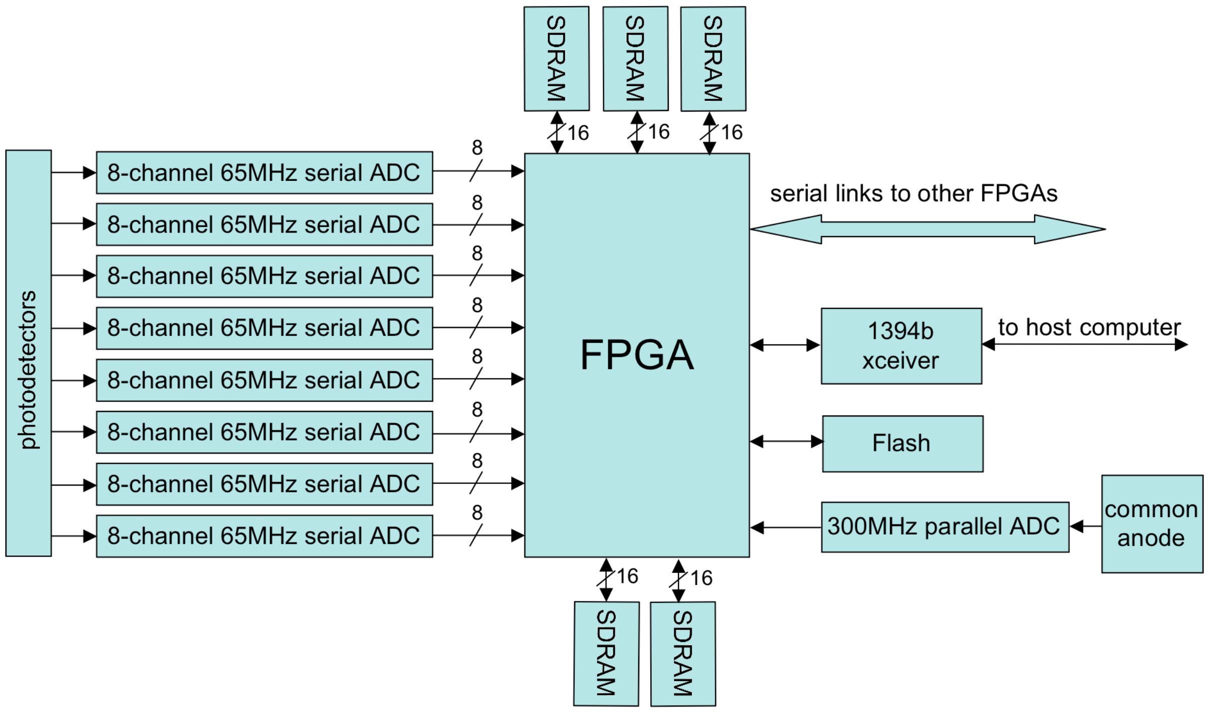


Figure 7.1. Block diagram of the major components of the data acquisition electronics for the MiCEs scanner.

In this architecture, each FPGA will process 64 channels from the photodetector. Each channel is sampled by a 12-bit 65MHz serial ADC. Serial ADCs are used because of I/O limitations. With 64 channels of 12-bit data, parallel ADCs would require 768 of the 976 user I/Os. The deserializers in the FPGA are limited to 1GHz, so that limits our ADCs to less than an 83MHz sampling rate. In order to improve the timing resolution of the system, a single 300MHz parallel ADC samples the signal from the common anode of the photodetector (see Figure 7.2). In chapter 4, we showed that a 300MHz ADC achieves a 3.8x improvement in timing resolution over a 65MHz ADC. It should be noted that this result is for a data set derived from individual pixels. Currently, it is not known whether the signals that arise from the common anode will have the same timing characteristics. Experiments with the common anode are ongoing.

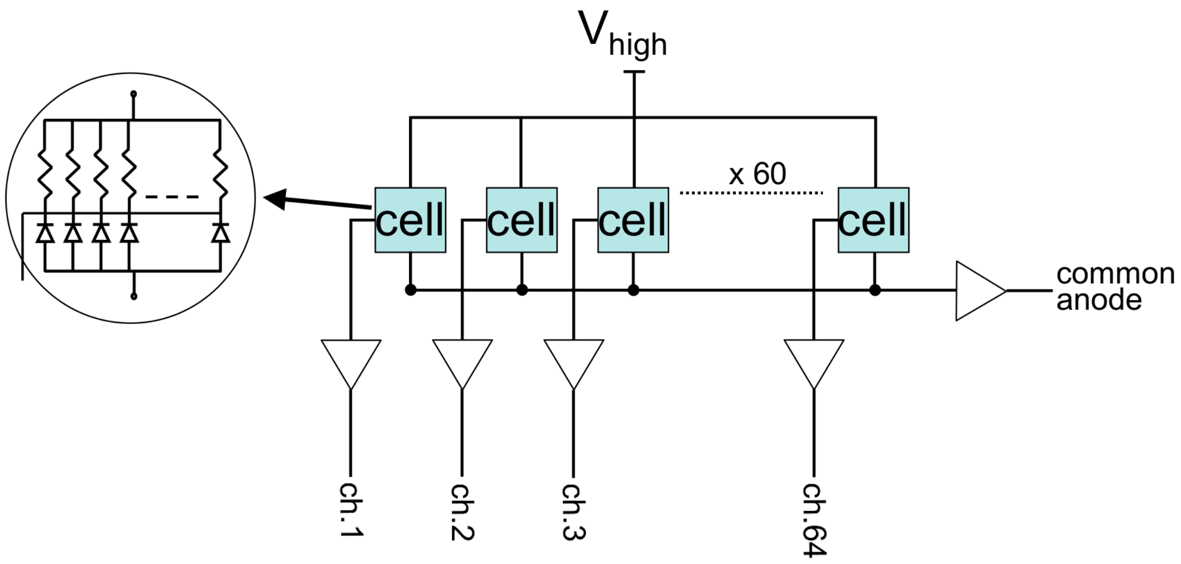


Figure 7.2. Illustration of the common anode that is connected to the output of all 64 cells in the MAPD array.

The SDRAM memory attached to the FPGA serves two functions. One set of memories is grouped together on a 32-bit bus to serve as the memory for the NIOS soft-processor that will be configured into the FPGA. The other set of memories is on a 48-bit bus and serves as a large characterization table for a statistical based positioning algorithm [28,29] that is used to determine where the photon interacted with the scanner.

A FireWire (1394b) transceiver is used to connect each FPGA to the host computer. The ring of our small-animal scanner consists of 18 detector modules, and each module is processed by a separate FPGA. The data that the FPGA produces for each event is sent to the host computer for image reconstruction via this FireWire bus.

The flash memory is utilized for the non-volatile memory to store some algorithm information during power down. For example, the reference pulse will be stored in the flash memory so that after power down the pulse discovery routine doesn't need to be run again. Likewise, the characterization tables for the statistical based positioning algorithm will be stored in the flash and transferred to the SDRAM and on-chip memory during power up.

7.2: Overall Signal Processing Architecture

The hardware and the signal processing algorithms of a PET scanner have a strong relationship. Sometimes the algorithm dictates the hardware requirements like the need for SDRAM for a statistical based positioning algorithm. Sometimes however, the hardware places limitations on the algorithms, as was the case in the timing circuit. In this case, the FPGA's method of computation places restrictions on the possible algorithms. The logic usage restrictions are in place because there are many computations that need to be placed in the FPGA and there are 64 channels that need to be processing in real-time. In addition to duplicate channels, there are also other tasks that need to be performed on the data. These other tasks include photon interaction location, pulse pile-up correction, baseline restore, a NIOS soft-processor, portions of a FireWire stack and other system applications.

All of these computations interact to provide the host computer the location, time and energy of the photon interactions with the scanner. The first step after a pulse is detected on a channel is to send the pulse through the pulse pile-up correction circuit. This will produce the energy and time stamp for this pulse (and any piled-up pulses). The energy from the pulse will be used by the photon interaction locating circuit. All of this information about the interaction is bundled up into a packet and sent to the FireWire buffer for eventual transmission to the host computer.

This processing chain for one channel is fairly straightforward, but processing all 64 channels provides a larger challenge. The simple solution of just duplicating the event detection and pile-up mitigation circuit 64 times will probably not be possible give logic constraints. (Note that the circuit for locating of the photon interaction point doesn't need multiple instances since it is locating one spot of interaction on the whole photodetector based on information from all of the channels). Instead, some buffering and multiplexing of detected events may need to occur in order to share pulse pile-up correction circuits between multiple triggers (Figure 7.3). In such a system, each channel will need to have a trigger circuit to detect pulses. Once a pulse is detected, it will be placed in a FIFO to be buffered. In the case where pulse pile-up correction will be used, enough samples will have to be buffered to collect the longest pileup event that will be processed. In addition to the pulse samples, the coarse time stamp of the first sample must be kept with the data, because it is impossible to know when the pulse/s will be removed from the FIFO. The number of pulse pile-up correction circuits will depend on the expected count rate of the detector over all 64 channels. While the processing circuits should be designed to handle the expected count rate, during times of elevated activity there is the possibility of missing some events in this system. If a buffer is full when a new event arrives, that event will have to be discarded.

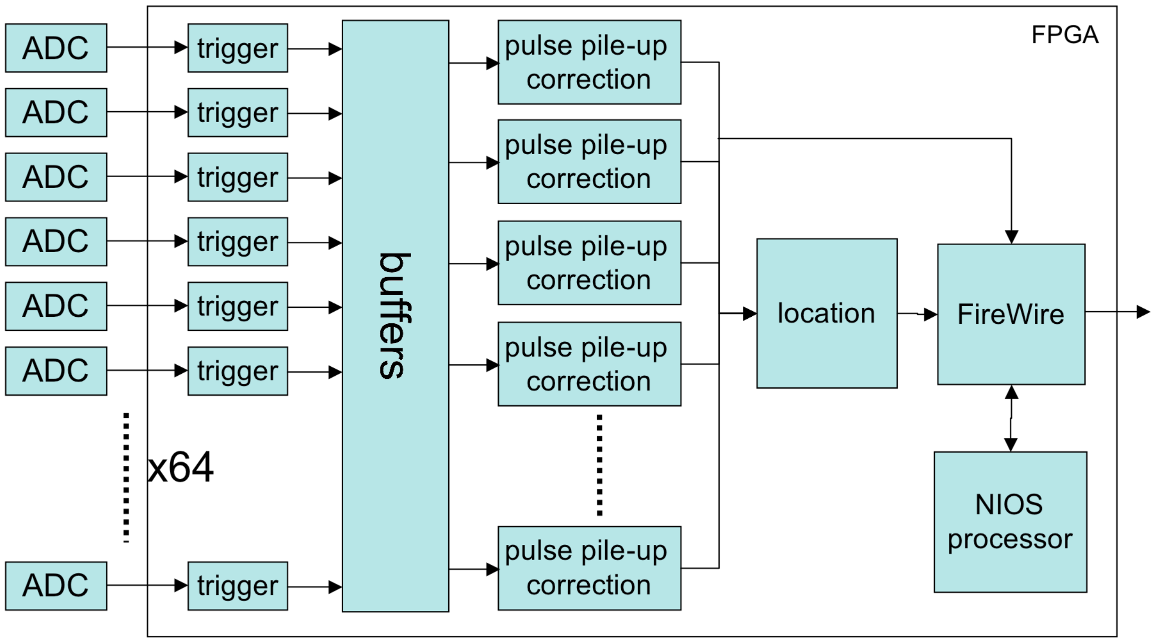


Figure 7.3. Illustration of the main parts of the pulse processing circuit inside the FPGA.

The buffers in Figure 7.3 are ambiguous because there are multiple ways to implement them that have various tradeoffs. One method is to have a FIFO attached to the output of each trigger to buffer the pulses. This architecture would have the easiest write control as the trigger only needs to check if the buffer is full before writing. The read control for the pulse pile-up correction circuits would be much more complicated. The read process would have to search for buffers with available data in a manner that assured all buffers get serviced with equal frequency. Otherwise, some buffers may fill faster than others. The other issue with this architecture is that it increases the chances of filling up a single buffer and therefore losing data. Since each buffer will be relatively small, a short spike in activity that is localized may result in a FIFO overflowing. The other end of the spectrum is to have one single big buffer into which all of the trigger circuits write. This would simplify the read process at the expense of the write control. Taking the first pulse set out of the buffer and sending it to an available pulse pile-up circuit would accomplish the reads. The writes would be more complicated because all 64 trigger circuits would be writing to the same FIFO, so an arbitrator would have to decide on who writes in the case of contention. It is also possible to lose data in this scheme because a trigger circuit will have to stall in the case that someone else is currently in the process of writing to the buffer. A third alternative that is a mix of the first two is to have one buffer for each pulse pile-up circuit and have a certain subset of the 64 trigger circuits feed each buffer. This has the easiest read protocol as the pulse pile-up circuit only has to check that the FIFO is not empty. The write control will be the same as the previous architecture with the single buffer, except fewer trigger circuits will be writing to the same buffer, so the likelihood of write collisions will be lower. Additionally, collisions can be lessened by strategically selecting which channels connect to which pulse pile-up circuits. For example, when a continuous slab scintillator is used, many pixels in the photodetector will receive pulses, but they will tend to be near one another. A scheme that spreads out neighboring pixels to separate pulse pile-up circuits will reduce write contentions into their buffers.

The above discussion assumed that pulse pile-up correction or timing would be required on each of the individual channels. There may be certain scanner configurations that only require timing and pulse pile-up correction on the common anode channel. Recall that the common anode channel is essentially a sum of all of the pulse channels, so it will have a higher incidence of pulse pile-up than any single channel. It may be possible to get the timing information just from the common anode channel, and only use the individual channels for locating the interaction. If the count rates are not too high, then there will be no need to perform pile-up correction on the individual channels. The energy can be calculated on each channel assuming no pile-up. In this architecture, there would probably be no need to buffer the data from the trigger circuits, as the processing requirements on each channel would be minimal.

Ultimately, the final arrangement of the algorithms developed in this thesis will depend on the configuration of the scanner as well as the functionality of the common anode circuit. If the common anode demonstrates good timing characteristics, then this will simplify the final FPGA design architecture. If however, it turns out that timing still needs to be performed on each channel, then a scheme like that shown in Figure 7.3 will need to be utilized. The big unknown about the common anode is whether the pulse shape will behave predictably enough from event to event to utilize in our algorithms.

Chapter 8 : Conclusions and Future Work

This dissertation presents the advantages and difficulties in creating an all-digital pulse processing chain for a PET scanner. Digital circuits have many advantages over analog circuits, which has led to the conversion of many analog functions into the digital domain. As the ADC sampling rates and the computational power of digital devices improve with technology trends, it becomes possible to convert more functions from the analog to the digital domain. These conversions are not necessarily direct translations however, since the strengths of the two domains are definitely different. Furthermore, different digital devices have different methods of computations that require careful targeting in order to achieve the best results. In this thesis we showed that, given the requirements of a research PET scanner, FPGAs are currently the best digital computing platform for this domain. Given this, we present a set of all-digital pulse processing algorithms for PET that are specifically designed to leverage the FPGA's strengths.

The objective of this thesis was to create a suite of digital pulse processing functions, targeted to FPGAs, to replace the analog circuits typically required for PET detectors. The hypothesis we started with was that using all of the samples in a pulse would give the algorithms more immunity to noise and therefore perform better than the analog circuit, which tend to rely on a single point on the leading edge of the pulse. It turns out that the analog algorithms were correct, because the best timing information is on the start of the leading edge. This indicated that there might be little room to improve over the analog circuit in terms of timing resolution. This held true, and the small improvements observed were the result of using multiple samples on the leading edge (at higher ADC sampling rates) and using a smart trigger. The big improvement is the switch from a custom analog circuit to a circuit built with commodity, off-the-shelf FPGAs while maintaining the precision. Considering that most PET systems already have a digital device to interface with the host computer, migrating the functionality to all digital platform will produce significant savings in circuit area.

Chapter 4 presents the notion of photon interaction timing and how it relates to the overall performance of a PET scanner. The timing resolution of a scanner is the most important factor in accurately determining coincidental photon pairs. The resolution is dependent on many factors. The hardware of the scanner (photodetector, electronics, etc.) effectively sets a lower limit on the timing resolution. The timing algorithm determines how close the system will get to this lower limit. The current state of the art in timing circuits is realized in the analog domain, which achieves timing resolution down to a couple hundred picoseconds. However, analog circuits are expensive to create, consume a lot of power and are inflexible. Digital circuits implemented in FPGAs on the other hand consume lower power, are easier to create and can be changed after implementation. While these characteristics makes FPGAs an attractive technology for the data acquisition and pulse processing hardware of a PET scanner, the digital algorithms need to achieve results similar to those of the analog circuits in order to make them a viable option. If the digital circuit severely underperforms, then there will be too much noise introduced into the final image to make it a reasonable solution. Given these performance requirements, an algorithm based on a pulse fitting routine was created and refined to leverage the strengths of the FPGA. The general idea of fitting a high-resolution reference pulse to the lower-resolution ADC pulse worked well, but it was obvious from the beginning that it couldn't be realized in a real-time implementation. The search portions of the algorithm (amplitude and time) were converted into real-time functions utilizing known parameters of the photodetector pulses. In this process, it was discovered that the leading edge of the pulse provided the most accurate timing information. This is exploited by the analog timing algorithms: The leading-edge detector and constant fraction discriminator both utilize data solely on the leading edge of the pulse.

Our digital algorithm achieves a timing resolution of about 3ns for a 65MHz ADC sampling rate, but it quickly improves as the ADC rate increases. In fact, at 500MHz, the timing resolution is about 750ps (this if for an older generation SiPM and should improve as newer generations improve), which is slightly better than the best analog algorithm. The slight improvement is a result of having a smart trigger that can determine if the first detected sample is noise or a part of the pulse.

Other digital timing algorithms have achieved similar results, but at the cost of significantly more resources or with the aid of external triggers. In an FPGA-based pulse processing system for PET, it is important to get the best performance with as little resources as possible. As chapter 7 illustrated, there are many parts to the pulse processing hardware, so the timing algorithm has a limited resource budget. The timing circuit presented in chapter 4 consumes limited resources and is focused on lookup tables, which make a very efficient implementation in FPGAs. In fact, a single timing circuit consumed less than 1% of the FPGA logic and memory resources.

While the timing circuit presented in chapter 4 achieves good timing resolution without using a lot of FPGA resources, there is a complication of creating an accurate reference pulse. The timing experiment with live streaming data from photodetectors illustrated this complication. The timing resolution for the FPGA experiment was significantly worse than simulations on the same data suggested it should be. This degradation was mostly due to an incorrect reference pulse. At lower sampling rates, simulations show that the timing resolution can be more than two times worse if the wrong reference pulse (a reference pulse from a different detector) is used. It turned out to be difficult to determine the shape of the pulse that the FPGA processed. Mathematical models don't fit the data very precisely, and modeling the shaping characteristics of the data-acquisition chain is difficult and unique for each setup. Using an oscilloscope to sample the pulses would be tedious to do for a whole scanner, and is also limited in its accuracy. The ADC, circuit board and FPGA have an influence on the shape of the pulses, but the closest the scope can get to an unsampled pulse is just before the ADC.

Chapter 6 solves this problem by creating an iterative FPGA routine that generates the higher resolution reference pulse from the same type of lower resolution data pulses it will later process during a scan. Along with the reference pulse, the routine calculates the other pulse parameters, such as average pulse energy and length, needed for the timing and other signal processing algorithms. The task consists of making a reference pulse that is defined about every 60ps from data that is sampled at ADC rates (65MHz to 1GHz). The most difficult step in this process is determining where the samples from the ADC pulse go in the higher resolution array that makes the reference pulse. This step attempts to determine what part of the pulse the ADC sampled in order to place it in the higher resolution array that will form the reference pulse. This is accomplished by normalizing the samples based on the area under the pulse and then determining how far the first sample is from the start of the pulse using a timing algorithm. After enough pulses are collected, the multiple samples per bin in the array are averaged and smoothed to generate a reference pulse. This process is then iterated two more times to produce the final pulse.

This result was plugged into the timing algorithm developed in chapter 4 to determine how well the routine did forming the reference pulse. The metric was to compare it to the timing resolution obtained by using a hand-tuned reference pulse that was formed using the pulses sampled at 25Gsps. The timing resolution for the reference pulses discovered in this algorithm matched that of the hand-tuned reference pulses, except for the simulation with a 65MHz ADC. At 65MHz, the timing resolution was 27% worse, indicating that the timing resolution at 65MHz isn't quite good enough to build a high-quality reference pulse. This is particularly true on the first iteration when a linear interpolation is used to perform the timing. If the 65MHz channels will be utilized for timing in the scanner described in chapter 7, then a higher order interpolation may be required.

Higher order interpolations can be difficult to efficiently program into an FPGA, but an efficient implementation is not a big concern for this algorithm since it is a tuning algorithm and only runs occasionally. The general procedure is to collect a bunch of pulses and then iteratively process them in order to make and refine a reference pulse. This leaves little opportunity for parallelism and involves a lot of control, but given its purpose, the main requirement of this algorithm is that it builds a high-quality reference pulse. It doesn't matter that it can't be built in real time, just that is built in a reasonable timeframe (few minutes). The only real-time constraint is the removal of the baseline and detecting the pulses. This is the same process that is performed on the front end of the timing routine, which has been shown to be capable of running in real time.

The reference pulse generated in chapter 5 can also be utilized to correct for pulse pile-up. Pulse pile-up is a complication that occurs in all PET scanners to some degree. The rate of occurrence is dependant on many factors, including the scanner architecture and the amount of tracer injected into the patient. The scanner presented in chapter 7 may have some architectural features that increase the likelihood of pulse pile-up, so correction may be important to preserve good timing and energy resolution. Pulse pile-up correction also allows more data to be collected for the same amount of tracer activity and scan time. For low count rates (100kcps) the timing resolution is 10-25% worse if no pile-up correction is used, while at higher count rates (1Mcps) it is almost 2x worse. For the rates at which we expect to run our scanner (~300kcps), the timing resolution degrades by 20-50%, depending on the ADC sampling rate. If the pile-up were just detected and discarded, the result would be the loss of almost 1/3 of the pulses at 1Msps. At 300kcps, the loss would equal only 10% of the pulses.

In order to collect and process as many pulses as possible, a pile-up mitigation routine was developed by adapting the timing routine. The first thing that was discovered was that the whole pulse is not needed to convert the area under the pulse to its amplitude. In fact, using the area from the start of the pulse to just past the peak for the normalization step achieves better timing resolution than using the sum of the whole pulse. This indicates again that the tail of the pulses contain a lot of noise and therefore degrades the timing resolution. It was also demonstrated that the tail of a pulse could be interpolated with the reference pulse as long as the start of the pulse (just past the peak) was free of pile-up. Given these results, it was determined that the cutoff between pile-up that can be corrected, and the pile-up that would be discarded, is just after the peak of the pulse (20% of the pulse length for the data set used in this thesis). However, there is a side effect of only using 20% of the pulse to estimate its area: The calculated area of a given pulse is somewhat dependent on the relationship of the first sample and the threshold of the trigger. Therefore, a correction factor has to be included in the area that is dependent on the voltage of the first sample.

In addition to reducing the amount of pulse used for amplitude normalization, other changes to the timing routine had to be made in order to process pile-up. The first addition is the peak pile-up filter to determine when the pile-up started before the 20% cutoff. Currently, the filter works by looking for too much energy in the first 20% of the pulse, which indicates the presence of two pulses. While this filter worked well enough to produce decent timing resolution, chapter 6 introduced some possible improvements to the filter. The other addition needed is a circuit to remove the pulse from the stream after they are processed for timing and energy. In this step, the reference pulse is used to interpolate the remaining 80% of the pulse and subtracts that out of the data stream from the ADC.

Simulations indicated that these additions to the timing circuit are effective at correcting pulse pile-up without degrading the timing or energy resolution a great deal. The timing resolution degrades by about 35% from 100kcps to 1Mcps. From 100kcps to 300kcps (our expected count rate) the timing resolution only degrades by about 10%. The difficult part of pile-up correction is differentiating between peak and tail pile-up. We accomplished this by using a two part filter that looks for too much energy in the first 20% of the pulse as well as a second rising edge after the peak of the first pulse.

We are in the process of creating an FPGA implementation of pulse pile-up correction. It is based on an extension of the timing algorithm developed in chapter 4. The baseline restore, trigger, and time lookup are identical and the area-to-amplitude normalization is the same computation except for the content of the lookup table. The major additions are the circuits that remove the pulse from the stream and the energy correction table. Both of these are essentially lookup tables, so an FPGA implementation will be fairly straightforward. Additionally, these "pile-up engines" will be cascaded so some control circuitry is added to manage which engine is currently processing the data stream.

The implementation of the pulse pile-up engine indicated that it will use about three times the resources of a single timing circuit for one pulse pile-up correction engine. Additionally, there is some degradation in the timing and energy resolution that is associated with the pile-up correction algorithm. Given this, should pulse pile-up correction always be used? When it comes to pulse pile-up in a data stream, there are three options. The pileup can be ignored by processing the data stream as if pile-up doesn't exist. The second option is to look for pile-up and reject it without attempting to separate the overlapped pulses. The final option is to implement a pile-up correction scheme like the one we presented in chapter 6 in an attempt to take apart any separable piled-up pulses in order to get the necessary information from them. It probably makes sense to at least detect and reject pulse pile-up because timing resolution degrades by 10-25% (depending on the ADC rate) at 100kcps if the stream is processed as if pile-up doesn't exist. This approach would only require the addition of a pile-up filter and wouldn't require multiple copies to be cascaded. The pile-up filter in this scenario can look for too much energy in the whole pulse while still processing with only 20% of the pulse, which will eliminate the issues encountered with the peak pileup filter.

Whether it makes sense to include a pulse pile-up correction circuit in the final design depends on the expected count rate of the system and the availability of resources. One circuit that is utilized for interaction location with a continuous slab crystal [27,28] requires more memory than is available on the FPGA, and a significant portion of the logic. The pulse pile-up circuit has a number of lookup tables that are implemented in memory as well. The question in this configuration is what would be the consequence of moving more of the event location memory off chip to make room for the pile-up circuit's memory? There are also some scanner configurations where the event location circuit won't be required, so there will be plenty of available resources on the FPGA. However, even when there are enough free resources, there may be instances when it may be a disadvantage to use. At 100kcps, less than 3% of the pulses are involved in pile-up, but correcting for these pile-ups degrades the timing resolution by about 5%. The question is which path introduces more noise to the final image? Losing 3% of the pulses reduces the signal-to-noise ratio, but so does increasing the coincidence window by 5% because of the timing resolution degradation. To determine which option results in the highest signal to noise ratio in the final image, a statistical analysis of the benefit of capturing more pulses versus the degradation of the timing resolution would need to be performed for the expected count rate. There is also a factor of scan time in this equation. PET scans are on the order of 20 minutes, so better efficiency at capturing pulses can lower scan times or reduce the needed tracer dosage. Given the reduction in timing resolution when pulse pile-up is not detector nor corrected (Figure 6.25) for even 100kcps, it is apparent that pulse pile-up should be detected and eliminated at the minimum. Beyond that, the decision to use pulse pile-up correction is a multi-variable question including desired count rates, FPGA resources available, and the need to collect more samples versus the need for the best timing resolution.

Taken as a whole, this thesis provides an insight into the future of PET data acquisition and pulse processing electronics. Digital signal processing is poised to replace analog circuits in the front-end electronics of PET scanners. The digital domain is certainly easier to work in and experiment with, but the digital circuits have generally trailed in performance for PET pulse processing. The sampling rate of ADCs and the processing power of FPGAs are improving at such a rate that the performance gap is rapidly disappearing. However, faster ADCs and better FPGAs alone are not enough to eliminate this gap. Developing digital circuits, especially ones constrained to a certain platform, requires a different design methodology than developing an analog circuit for the same function. This thesis illustrates this design process and demonstrates that the performance gap between analog and digital signal processing can be eliminated. We were able to utilize the strengths of the FPGA to achieve good timing resolution with a fairly compact circuit. In the same manner, the timing algorithm was extended to handle pulse pile-up. The integration of timing and pulse pile-up correction reduces the overall logic requirements. Finally, the ability to reconfigure the FPGA was utilized to create a tuning algorithm to discover the necessary pulse information for digital pulse processing. This parallelizes the information discovery process across the whole scanner, and removes most of the manual input.

The disadvantage of this pulse-processing suite is the heavy reliance on one piece of data, the reference pulse. This in turn relies on the pulse parameter discovery routine presented in chapter 5. If the reference pulse is inaccurate because of the discovery routine, then the timing resolution and the accuracy of the pile-up correction will suffer. Likewise, if the pulses from the photodetector don't have a consistent shape, these algorithms will under perform. The work in this thesis relies on a pulse that has little variance in its pulse shape from event to event. This hypothesis held true for the data sets utilized in this thesis, but there is no guarantee that future photodetectors will behave similarly. We also assume that the best timing information is on the leading edge of the pulse. This also may not hold true for future photodetectors.

Furthermore, there is a bit of a synthetic nature to the data set utilized in this thesis. The data set is composed of real pulses from a photodetector, but each pulse was a single event that was screened for its goodness. These single pulses were then stitched together to make the pulse streams required for the different tests. There may be some aspects of this process that don't match the characteristics of a real pulse stream from a photodetector. Particularly when it comes to pulse pile-up, there may be some artifacts in the baseline that wasn't captured in our simulation. In order to truly evaluate these algorithms, a rigorous test with a working PET scanner will be required. I expect that the core of the algorithms will perform similarly to the simulations, but there may need to be some additional logic to handle the extra pulses seen from background radiation and scatter, as well as different noise patterns.

Like many other analog circuits, PET data acquisition seems poised to transition to the digital domain. Future research will determine whether the algorithms presented in this thesis are the solution. This includes testing the algorithms in a working scanner as well as other possible solutions for event timing and pulse pile-up correction. While there have been other solutions proposed, I feel that currently, the algorithms presented in this thesis represent the best solution to date in terms of performance and logic requirements. Additionally, it seems that any future solution will be based on the principals discovered in this thesis: the most accurate timing information is contained on the leading edge of the pulse, and the variations in amplitude must be accounted for.

Bibliography

[1] R. Fontaine et al., “Timing Improvement by Low-Pass Filtering and Linear Interpolation for LabPET™ Scanner” *IEEE Trans. Nuclear Science,* 2007, vol. 55, issue 1, part 1, pp. 34-39.

[2] M.D. Fries, J.L. Willaims, "High-precision TDC in an FPGA using a 192-MHz quadrature clock," *IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 2002, pp. 580-584.

[3] J. Imrek et al., "Development of an FPGA-based data acquisition module for small animal PET," *IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 2004, pp. 2957-2961.

[4] C.M. Laymon et al., “Simplified FPGA-Based Data Acquisition System for PET,” *IEEE Trans. Nuclear Science,* vol. 50, no. 5, 2003, pp. 1483-1486.

[5] M.D. Lepage, G. Leger, J. Cadorette, J.D. Leroux, M. Otis, S. Rodrgiue, R. Lecomte, "FPGA/DSP-based coincidence unit and data acquisition system for the Sherbrooke animal PET scanner," *IEEE Nuclear Science Symp. and Medical Imaging Conf.*, 2000, pp. 173-175.

[6] W.W. Moses, M. Ullish, “Factors Influencing Timing Resolution in a Commercial LSO PET Scanner,” *IEEE Trans. Nuclear Science,* vol. 43, no. 1, 2006, p. 78-85.

[7] T.K. Lewellen, A.N. Bice, K.R. Pollard, J.B. Zhu, M.E. Plunkett, “Evaluation of a clinical scintillation camera with pulse tail extrapolation electronics”, *J. Nuclear Medicine,* 1989, vol. 30, pp. 1544-1558.

[8] J.S. Karp, G. Muehllehner, D. Beerbohm, D. Mankoff, "Event localization in a continuous scintillation detector using digital processing," *IEEE Trans. Nuclear Science,* 1986, vol. 33, pp. 550-555.

[9] J. Liu et al., “Real Time Digital Implementation of the High-Yield-Pileup-Event-Recovery (HYPER) Method,”*IEEE Nuclear Science Symp. Conf. Record (NSS/MIC)*, 2007, pp. 4230-4232.

[10] T. Lewellen, J. Karp, *Emission Tomography,* San Diego: Elsevier Inc., 2004, pp. 180.

[11] 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.

[12] Frach et al., "The Digital Silicon Photomultiplier – Principle of Operation and Intrinsic Detector Perfromance," *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC),* 2009, pp. 1959-1965.

[13] Degenhardt et al., "The Digital Silicon Photomultiplier – A Novel Sensor for the Detection of Scintillation Light,", *IEEE Nuclear Science Symp. Conf. Record (NSS/MIC),* 2009, pp. 2383-2386.

[14] A. Alessio, unpublished presentation.

[15] A. B. Brill, R. N. Beck, *Emission Tomography,* San Diego: Elsevier Inc., 2004, pp.25.

[16] H. Lim, J. Park, "Comparison of time corrections using charge amounts, peak values, slew rates, and signal widths in leading edge discriminators," *Review of Scientific Instruments,* 2003, vol. 74, no. 6, pp. 3115-3119.

[17] M. Streun et al., "Coincidence Detection by Digital Processing of Free-Running Sampled Pulses," *Nuclear Instruments and Methods in Physics Research Section A,* vol. 487, 2002, pp. 530-534.

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

[19] Altera Corp. "NIOS II Processor Literature" http://www.altera.com/literature/lit-nio2.jsp

[20] G.F. Knoll, *Radiation Detection and Measurement,* New York: John Wiley & Sons, Inc., 2000.

[21] 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.

[22] 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.

[23] W.H. Wong, H. Li, "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.

[24] M. Schmand et al., "Advantages using pulse shape discrimination to assign depth of interaction information (DOI) from a multilayer phoswich detector," *IEEE Trans. Nuclear Science,* vol. 46, no. 4, pp. 985-990.

[25] T.K. Lewellen et al., “Design of a Second Generation Firewire Based Data Acquisition System for Small Animal PET Scanners,” *IEEE Nuclear Science Symp. and Medical Imaging Conf. Record,”* 2008, pp. 5023-5028.

[26] J. Pasko, S. Hauck, “FPGA Implementation of Pulse Pileup Resolution for the MICES PET Scanner” UW Technical Report, UWEETR-2011-0002.

[27] T.K. Lewellen et al., “Evolution of the Design of a Second Generation FireWire Based Data Acquisition System,” *IEEE Nuclear Science Symp. and Medical Imaging Conf. Record,”* 2010.

[28] Johnson-Williams, N. G., Miyaoka, R. S., Li, X., Lewellen, T. K., Hauck, S., "Design of a Real Time FPGA-based Three Dimensional Positioning Algorithm", *IEEE Trans. Nuclear Science*, 2011, vol. 58, issue 1, part 1, pp. 26-33.

[29] N. Johnson-Williams, “Design of a Real Time FPGA-based Three Dimensional Positioning Algorithm,” M.S. Thesis, University of Washington, Dept. of EE, 2009.