Time division and the Effects of Noise Inside of Straws and a Case Study on Analog to Digital Converters and the Maximum Voltage Distribution of Electrons and Protons

Daniel Kulas

Bethune-Cookman University, Computer Engineering

Supervisor: Vadim Rusu

Mu2e

Abstract

To accurately determine the position of a particle passing through the straws of the Mu2e tracker, we can exploit the fact that we can measure how long it takes for a signal to reach the pre-amplifier circuits at each end of the straws. However, once one takes into account of the noise on the sensing wire, the time division measurements can get slightly off, thereby producing a location of the particle that is different from where it actually passed through the straw. Research was done to see how much does noise affect a signal assuming the particle passed through the middle of the straw. Also, when viewing the maximum voltage distributions on electrons and protons signals, there is a small overlap of the voltages between these two signals, thereby making it difficult to determine what particle is which. Simulations of different analog-to-digital converters were carried out to determine which will minimize the potential for error labeling which signal is which.

Introduction

Muons have a very short lived life with an average life span of 2.2μs. Once they decay, thy typically decay into an electron and two neutrinos, the tau-neutrino and the electron-neutrino with the mass of the muon being distributed throughout the three particles. However, it is theorized that a muon can decay directly into an electron. Mu2e is looking to see if a muon can decay directly into an electron with a mass of 105MeV. Mu2e will employ the use of use drift chambers inside of the tracker to determine if this decay of a muon to an electron is even possible.

A drift chamber is essentially a tube filled with a mixture of gas with a high voltage wire running through the center of the tube . Once a high energy particle passes through the straw, it ionizes the gas inside. Due to the high voltage wire in the center of the drift chamber, the electrons generated by the ionization get pulled towards this wire.

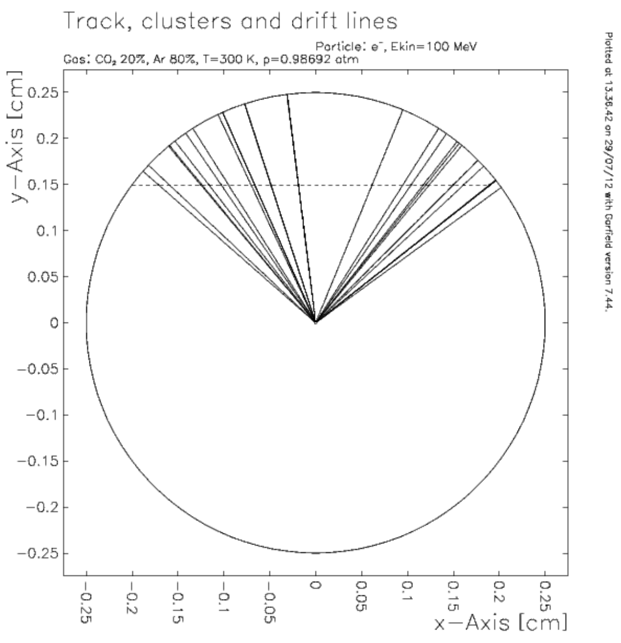


Figure 1, model of a 2D drift chamber generated by Garfield. The horizontal line represents the particle passing through the drift chamber. The solid line going towards the center represents the electrons formed during ionization and getting pulled by the high voltage wire.

As it gets pulled into the center, it bumps into other gas molecules along the way which can cause them to get ionized as well. This effect is called avalanching. As the electrons hit the wire, the current it produces flows down the drift chamber and is picked up by the electronics to detect the signal [1]. This process is simulated, in Figure 1, in a program used throughout the research. These drift chambers, hereby referred to as straws, employed in this experiment were to be used to detect the presence of particles passing through it.

In the Mu2e experiment, straws are used inside of the tracker to reconstruct the trajectory of the particles that flow through it.

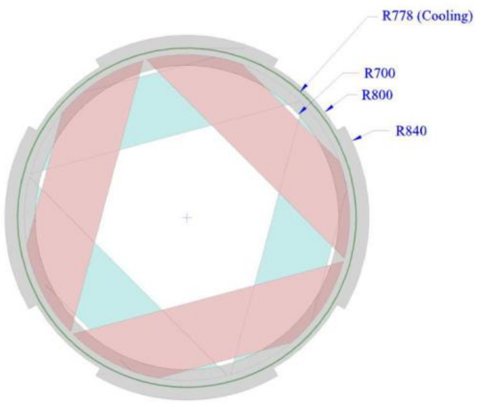


Figure 2 A cross section view of the tracker [3]

In Figure 2 the blue and red diagonals are arrays of straws lined side by side. With several of these panels lined up, they form the tracker which is shown in Figure 3. The straws are set up in these diagonal shapes due to the fact that charged particles flow in a helix. The particles will flow through the straws and a signal will be generated in each straw that they pass through.

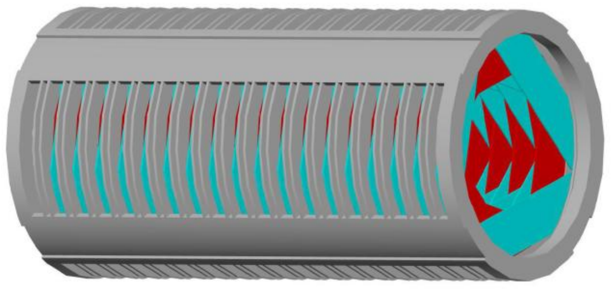


Figure 3 The tracker [3]

Since charged particles have a helix trajectory, the straws line up near the edge of the tracker rather than being placed through the entire volume of the tracker.

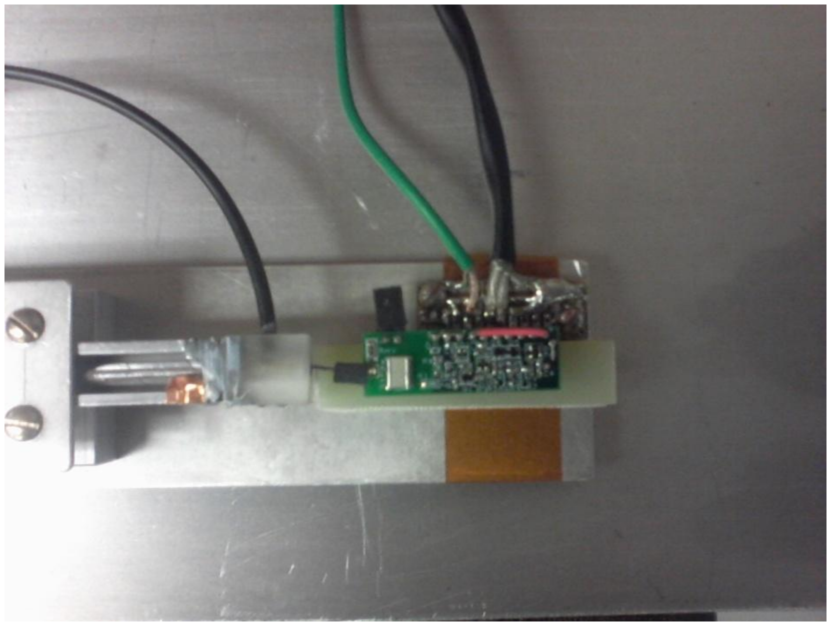


Figure 4 The pre-amplifier circuit board at one end of the straw

How can we reconstruct the path of the particles flowing through the straws of the tracker? Since there are pre-amplifier circuits at each ends of the straw, we can detect how long a signal takes to reach these circuits. This is referred to as time division. If a particle were to pass directly through the center of the straw, we should see that the signal is detected on both pre-amplifier circuits at the same time. If a signal were to pass on the left side of the straw, we should see the pre-amplifier circuit on the left end detect the signal before the pre-amplifier circuit on the right end of the straw. With this information we can then accurately determine where exactly the particle entered the straw. However, this assumes that there is no thermal noise on the system. Since this isn't a perfect world where we can ignore such noise, we have to research the effects it can have on the system. As it turns out, noise can pose a large potential for error when reconstructing the trajectory of the particles.

The pre-amplifier circuit doesn't allow for the output signal to be read into a computer for processing as the use of the analog-to-digital converter (ADC) is required for that. The ADC is able to take an analog signal and discretize it into a series of bits that a computer can interpret. When looking at the maximum voltage produced by electrons and protons, without applying an ADC on the signal, there is a small overlap of voltages as shown in Figure 5.

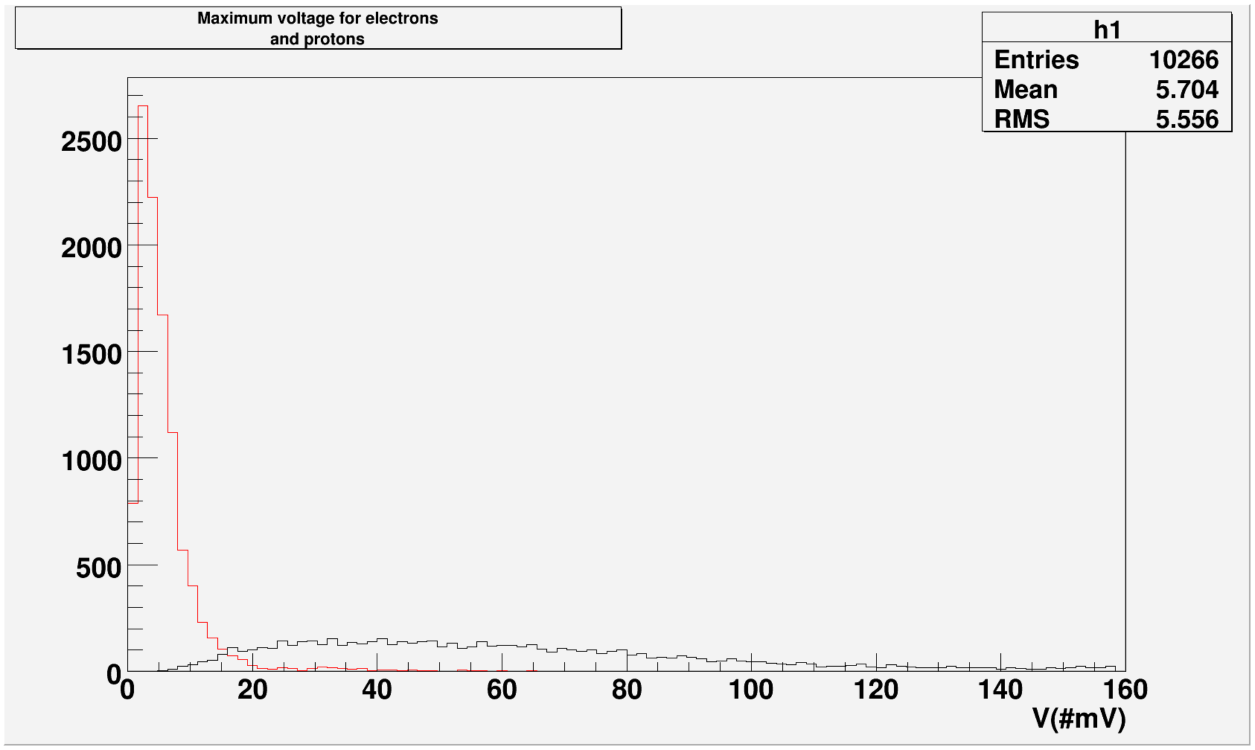


Figure 5 The maximum voltage values for protons (in black) and electrons (in red). Note the overlap of voltages around 0mV to 20mV

If we were to just look at raw data from the ADC, it's difficult to determine which signal came from which particle. Therefore, a case study was carried out to determine which ADC would be best to figure out which signal came from a particular particle. We adjust the parameters of the ADC ranging from the frequency of it to how many bits the ADC will be able to record.

Approach

Three main programs were implemented before we could start to run any sort of data analysis. To simulate the straws, the program Garfield was employed. Garfield is an open-source program from CERN that allows for simulations of 2D and 3D drift chambers. Ngspice, which is an open-source version of SPICE, was used to simulate the pre-amplifier circuit used at the ends of the straws and was used for the ADC study. Finally, ROOT, which is also an open-source program from CERN, was used for the data analysis.

Garfield was fed in a file that mimics the environment the straws are in. Parameters were set in this file to help generate data that would be expected inside the straws, such as adjusting the gas concentration inside of the straws, the track the particles enter and exit the straw, the energy level of the particles, and many others. See Appendix 1-A to view the source code for Garfield.

For our time division study, we generated 20,000 signals of electrons and gamma rays. We simulated these signals in a straw with a radius of 2.5mm, 80% argon and 20% CO2 gas concentrations, a 2 dimensional straw, 100MeV electrons and 6KeV gammas. The outputs were passed over to ROOT for analysis. Since Garfield doesn't simulate any effects of noise, this had to be done in ROOT. To simulate thermal noise, a simple equation was employed to calculate the noise current on the signal.

$$i_n = \sqrt{\frac{4k_B T\Delta f}{R}} \quad 1)$$

Where i_n is the current, k_B is the Boltzmann constant, *T* is the temperature, Δf is the noise bandwidth, and *R* is the resistance. The temperature is set at 300K, Δf is user defined in the program written for ROOT, and the resistance is at 300ohms. The value generated by this equation is randomly added on to the raw data based on a Gaussian distribution. The signals were then passed through a 100MHz or a 200MHz low-pass filter to smooth out the signals. To determine the Δ*t* of the two signals a measurement was taken at the point where the signal first crossed a threshold. A threshold was implemented to help determine if a signal was present and also to help avoid noise that would be more pronounced on smaller signals. If you apply the threshold too low, you dwell in the area where noise is prevalent and can get ugly results. If the threshold is too high, you can potentially miss signals all together.

Thresholding was done in two ways. First, a fixed threshold was applied on each signal that was based on the noise current on the signal. In the case where a user inputted a noise bandwidth of 100MHz, the resulting current noise will result to 0.0743μA. A histogram was generated to find the maximum current produced by noise. The maximum value that was found by generating this histogram was to be the threshold to set. This fixed threshold changes based on what the user inputs for their noise bandwidth parameter. In the case of a noise bandwidth of 100MHz, this threshold was set at 0.21μA. The other way thresholding was done was by taking the maximum value of the signal and dividing that by two. This would make the threshold well away from most of the effects of noise and would be a good spot to check the time it takes to cross this threshold. In the event that this new threshold set is lower than the fixed threshold, the program will revert back to the fixed threshold value. Since all we want to see currently is how noise affects the signal, and not to produce a full scale simulation of the system, this implementation is good enough for studying the effects of noise.

For the ADC case study, a new set of data was used. In the time division study, Garfield produced data randomly giving an equation set in the Garfield source file to determine where the particle entered and exited on a 2D straw at a fixed energy level. This new data set provided the coordinates of a particle's entrance and exit for a 3D straw and the energy level for each particle. This data was passed over to the Garfield file and produced data based on the input file.

Once Garfield gave an output, this data was then passed over to ngspice to simulate the pre-amplifier circuit. The ngspice source file can be found in Appendix 1-B. Ngspice then generated new data and was passed to ROOT for analysis. In ROOT, the ADC was simulated by randomly taking a sample at 50MHz and 100MHz and on a 4bit ADC or 8bit ADC. A cutoff voltage was set at 15mV. Any signal over this voltage was pushed to an overflow bit and wasn't used in the results. Next we wanted to record 95% of the electrons passed through the ADC. This is to say, out of all the values recorded on the histogram

Results

Time division measurements

For our time division measurements, we passed the signals through a low-pass filter; one at 100MHz and at 200MHz with a 100MHz noise bandwidth resulting in a noise current of 0.0743μA. The low-pass filters were used to remove high frequency signals on the system thereby reducing noise and smoothing out the signal. Here, in Figure 6, we see one of signals produced by Garfield.

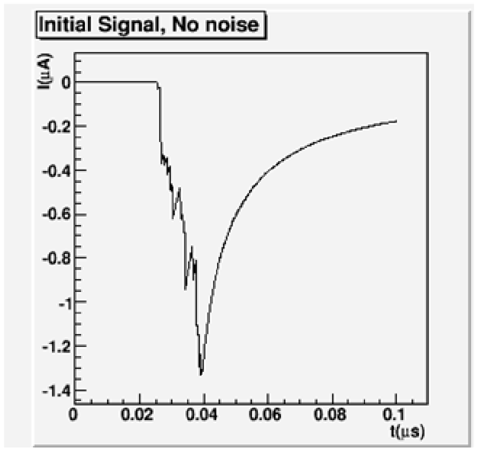


Figure 6, Electron signal with no noise and no filtering.

This signal has no noise added on or passed through any low-pass filter. This was used as a reference when analyzing the signals with noise and filtering applied. In Figure 7, we see the two signals that the pre-amplifier circuit would initially detect.

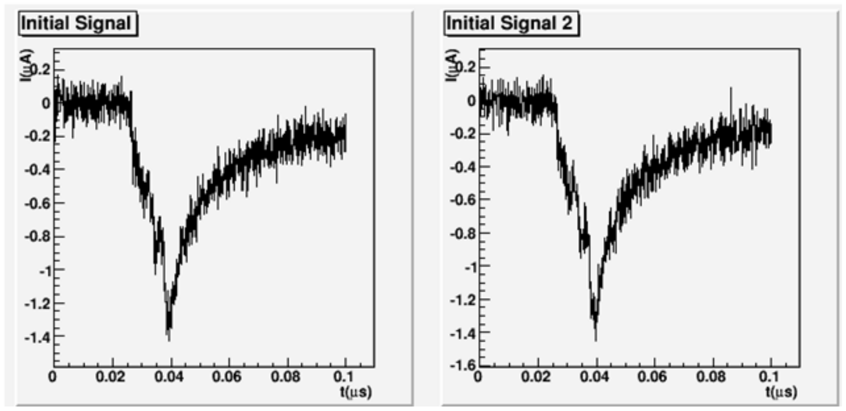


Figure 7, Initial signal with noise added with 100MHz noise bandwidth. This results in a noise current of 0.0743μA.

These signals have noise applied and with no low-pass filtering applied. These signals then were put through a low-pass filter and in figure 8, we see the resulting output.

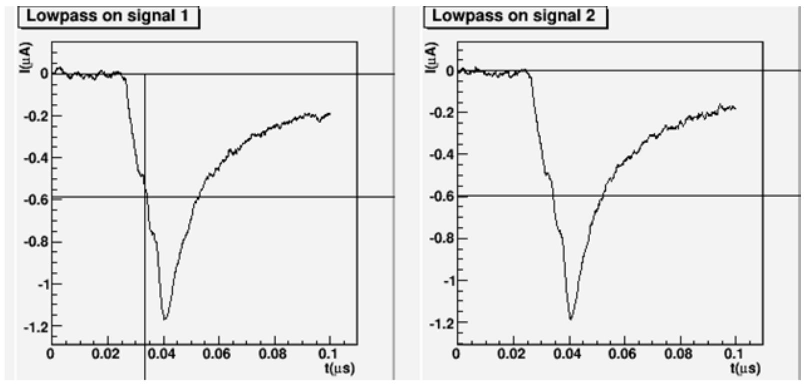


Figure 8, Initial signal with noise and a 100MHz low-pass filter applied. Horizontal line represents the threshold and the vertical line represents the time at which the ADC fired.

There are two things to take note of in these graphs. The horizontal line running across the two graphs represents where the threshold was set. Again, these thresholds were set based on the maximum current of the signal divided by two. The vertical line was used to represent where the ADC would fire and record the current of the signal. This was just used as a visual aid for the next portion of the study and serves no purpose in time division in its current form. When making these graphs, useful information is outputted to the terminal for more thorough analysis. This information included where the threshold was set, at what time did the signal crossed this threshold, the current of the signal at this time, and the Δt between the two signals.

Histograms were generated to record the maximum current produced by each signal and to record Δt of the two signals. Four histograms were produced for electrons and four histograms for gammas: two histograms for 100MHz low-pass filter on electrons, two histograms for 200MHz low-pass filters on electrons, and the same thing for gamma-rays. The resulting histograms were plotted on top of each other to bring out the differences between the two different low-pass filters.

Figure 9 shows the time division on the electrons at 100MHz in red and 200MHz in black.

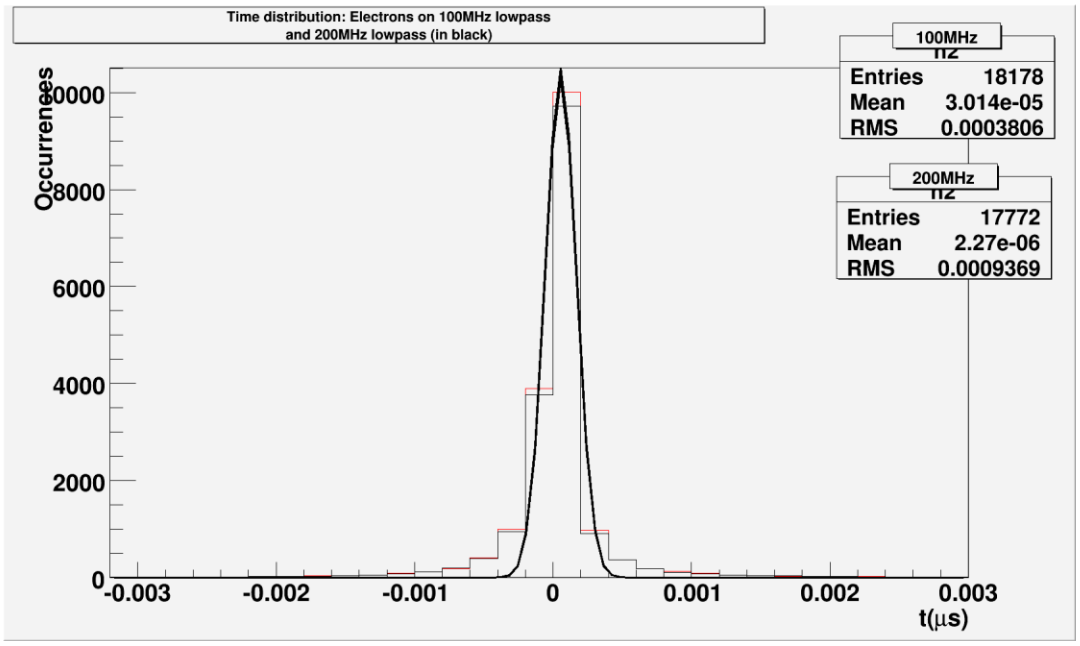


Figure 9, Time distribution of electrons. Red line = 100MHz low-pass w/ 100MHz noise bandwidth the Black line = 200 MHz low-pass w/ 100MHz noise bandwidth

There appears to be a very small difference between using the two low-pass filters, however Mu2e relies on how accurately data can be measured. When dealing with a timescale of 100 ns, 1 ns can have a major impact on the reconstruction of the particle's trajectory. One thing to point out is the fact that you lose signals with the current implementation of thresholding. There were 20,000 signals produced yet only 18178 signals were detected on 100MHz low-pass filters and 17772 signals were detected on 200MHz low-pass filters. This is because some signals produced by Garfield were very small; some signals were in the 0.12uA range. The fixed threshold set at a 100MHz bandwidth is at 0.21uA. So the signals below this threshold get tossed out. If one looks at the graph of the signal with a maximum current of 0.12uA, one can't really distinguish between the signal and the noise, even after filtering, without making reference to the original clean signal. So even though we lose data, if we didn't apply thresholding to the signal we could alter our data vastly. Figure 10 shows such a graph.

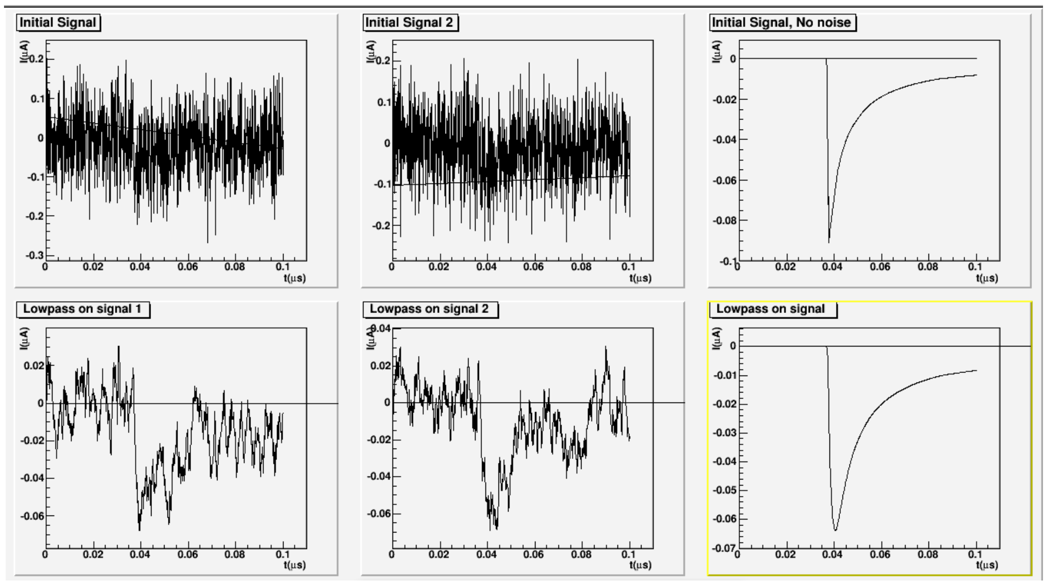


Figure 10 The original signal with no noise peaks around 0.09uA. Once noise is applied and the low-pass filter removes high frequency values, we are left with two completely different signals that don't represent the original signal at all.

The maximum currents produced by the electron signals at 100MHz and 200MHz, as shown in Figure 11 also shows a slight change in the currents. Ideally, we want signals that don't dwell in the realm of noise and the 200MHz low-pass filter appears to have signals 10% stronger than the 100MHz low-pass filter. With a mean of around ~1µA on 200 MHz low-pass filters compared to ~0.90µA, 200MHz appears to be the better choice.

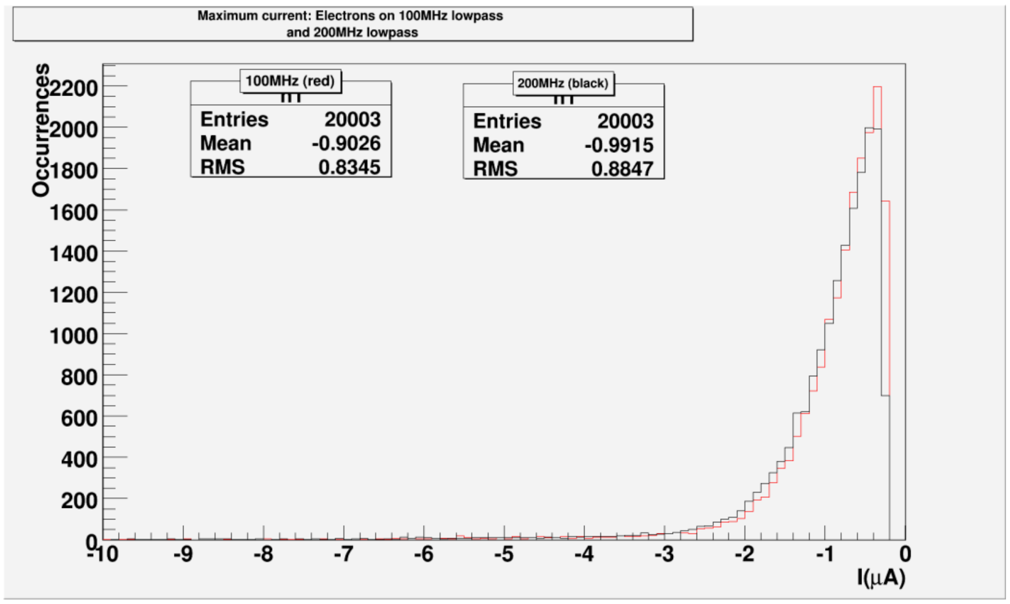


Figure 11 The red line represents electron signals passed through a 100MHz low-pass filter and the black line represents electron signals passed through a 200MHz low-pass filter.

The time division results when comparing 100MHz low-pass filter with the 200MHz low-pass filter seem to favor the 200MHz low-pass filter as show on Figure 12. The mean on the 200MHz low-pass is lower than the mean on the 100MHz low-pass suggesting a smaller tail on the Gaussian distribution. There is, however, the problem that a 200MHz low-pass filter loses 406 signals. Do we want to give up data in favor of better results? We lose about ~11% of the signals on a 200MHz low-pass filter whereas we lose ~9% of the signals on the 100MHz low-pass filter. One thing to take into account is that the low-pass filter only accounts for one part of the pre-amplifier circuit. So results can still vary once the signal is measured through the circuit.

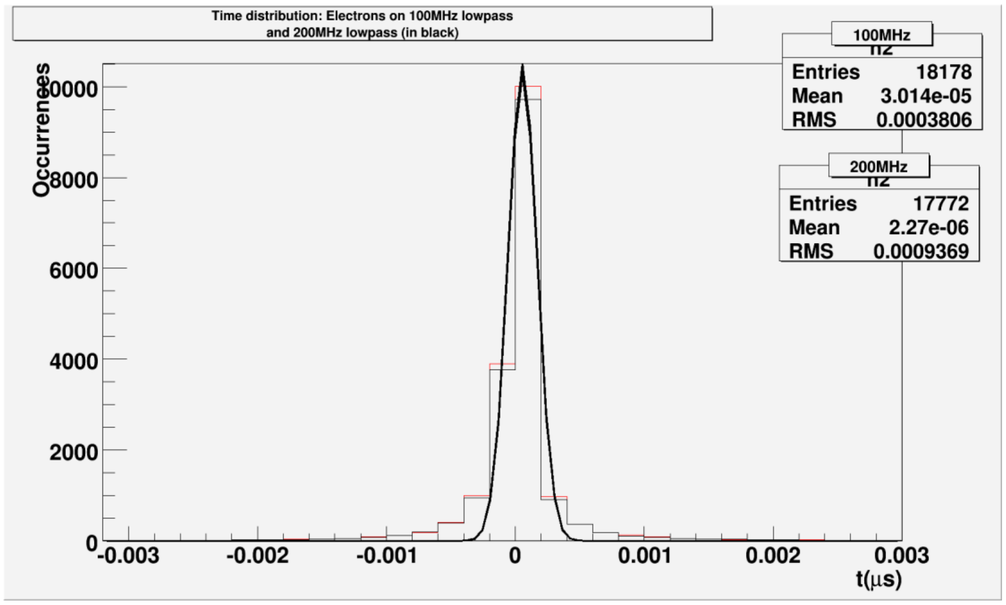


Figure 12 The time division results on a electron signals passing through a 100MHz low-pass filter (in red) and through a 200MHz low-pass filter (in black)

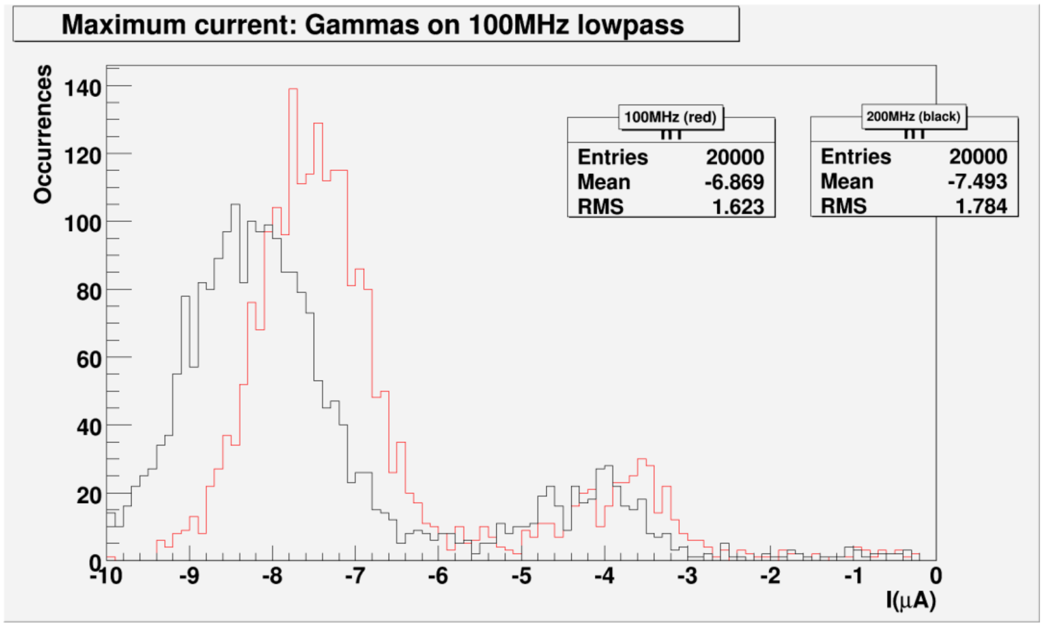


Figure 13 The red line shows gammas passed through a 100MHz low-pass filter and the black line shows gammas passed through a 200MHz low-pass filter.

Figure 13 shows the maximum currents produced by gamma-rays, and as seen, gamma-ray currents measure from 1uA to 10uA, with most of these signals averaging around 7µA to 8µA on a 100MHz low-pass filter and around 8µA to 9µA on a 200MHz low-pass filter. As it turns out, when analyzing the time division histogram, these larger signals tend not to be affected too much by noise. Figure 14 shows this result.

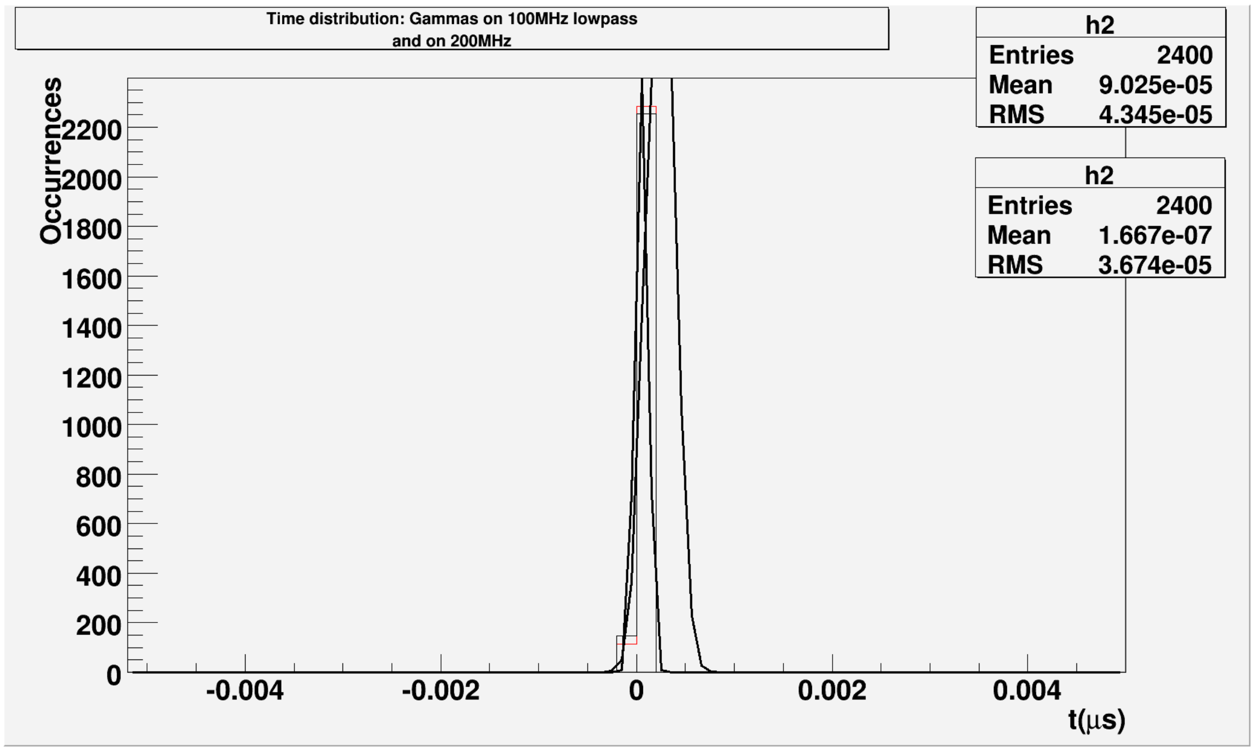


Figure 14 This is the time division result on gamma-ray signals passed through a 100MHz low-pass (in red) and 200MHz low-pass filter (in black). [REPLOT TO INCLUDE GAUSSIAN LINE]

How does this show why gamma-rays aren't really affected by noise? Due to the higher currents generated by gamma signals, they aren't affected by noise nearly as much as electrons are. First, electron signals tend to average out around ~1µA. The noise current at a 100MHz noise bandwidth produces a current noise around 0.0743µA. Noise has a greater affect on the smaller signals. Compare the electron signal in Figure 8 with the gamma signal in Figure 15.

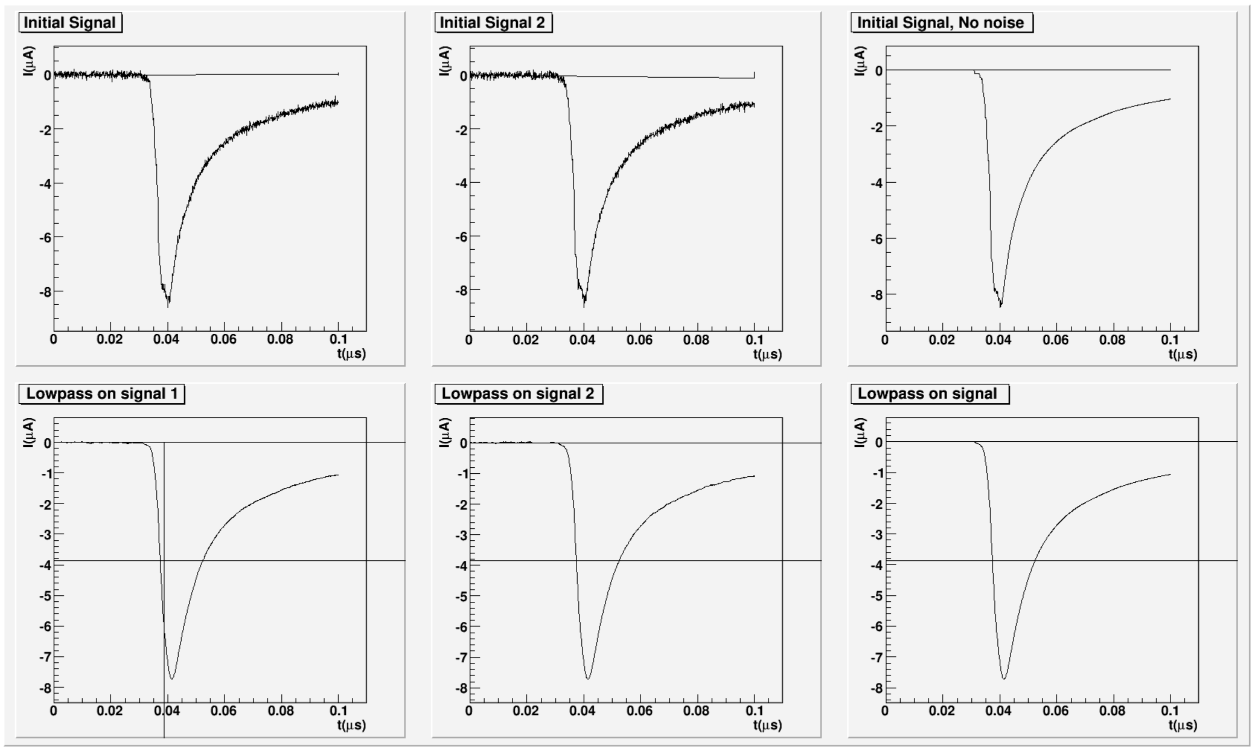


Figure 15 A gamma-ray signal

Most gamma signals tend to have high currents and sharp spikes up to their maximum current. Noise doesn't affect these signals as much as the smaller electron signals. To prove this claim, I took the gamma signals and reduced their current down to the same range as electrons. The histogram in Figure 16 shows this change. It can be seen that the gamma-ray signals look much like the electron histogram signals.

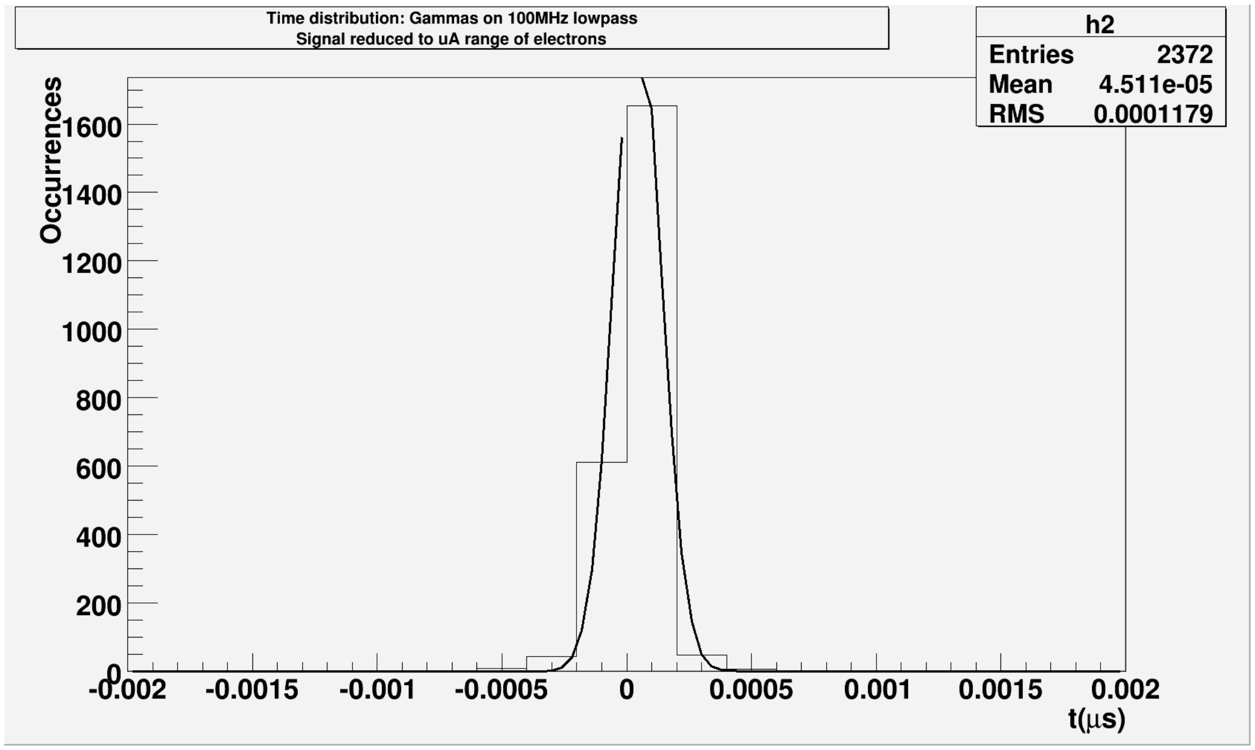


Figure 16 A factored down gamma signal time division histogram closely resembles the electron time division histogram in Figure 12.

Based on analysis, it can be said that applying a 200MHz lowpass filter to the signals will help preserve the signal strength and will help improve time division resolution. However, noise still prevents a problem even after efforts to reduce its presence. Having a particle pass through the straw closer to the left pre-amplifier circuit, noise can potentially cause the signals to reach the ends at the same time making the particle appear to cross directly through the middle of the straw. More research is required to explore for new ways to overcome this effect.

ADC Case Study

In our ADC case study, a new data set had to be generated to mimic the conditions of a particle passing through a 3D straw. To do this, two files were provided that contained entry and exit points of electrons and protons passing through the straws along with their energy levels. A script was written up to process these files and produce outputs that can be passed into ROOT for analysis. This script read the file in line by line, passed in parameters to Garfield, which generated data that was then passed to the pre-amplifier circuit in ngspice, then those outputs from the entire file were compiled into one larger file which was then passed to ROOT. This script can be viewed in Appendix 1-D. For analysis, there were 10,278 electron signals and 6,835 proton signals all together to be passed through the ADCs. We first ran the signals through a 50MHz 4bit ADC then through a 100MHz 4bit ADC. Figure 17 shows the result.

.

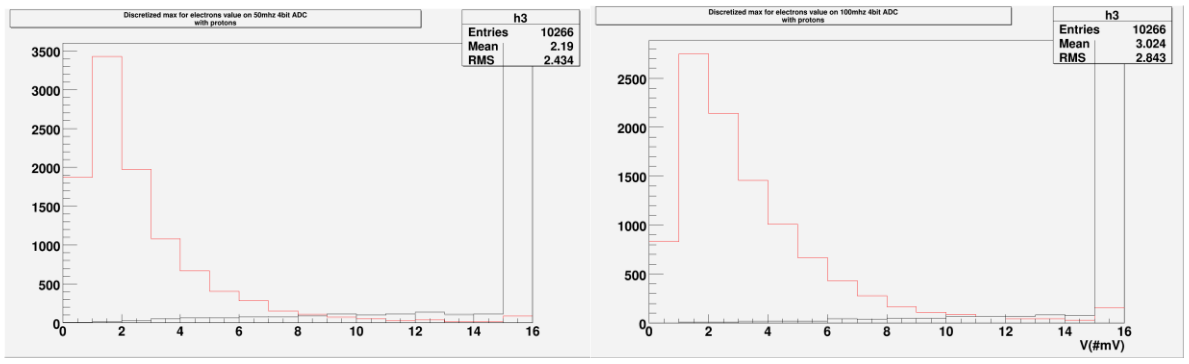
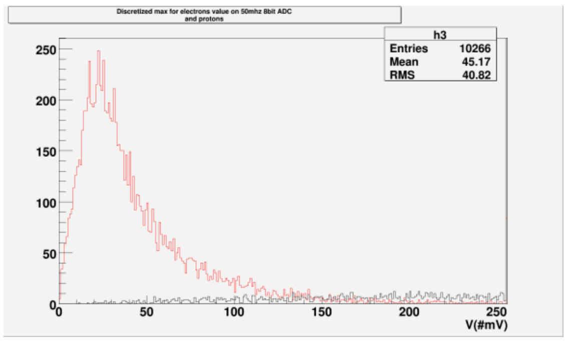


Figure 17 50Mhz 4bit ADC and 100Mhz 4bit ADC. The bins on the right most side of the histogram is the overflow bit. Any signal over the 15mV cutoff point was placed in this bin. The red line are electron values and the black line are proton values.

Then finally, the signals were passed through a 50MHz 8bit ADC and a 100MHz 8bit ADC. Figures 19 show the resulting histograms.



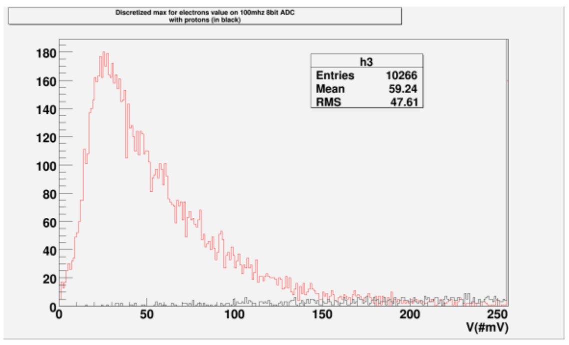


Figure 19 50Mhz 8bit ADC. The red line are electrons values and the black line are protons values
Figure 20 100Mhz 8bit ADC

ADC Type	Bin	5% cutoff	Rejection of Protons
100MHz 8bit	157	4.97%	96.95%
	156	5.08%	96.96%
50MHz 8bit	125	4.96%	95.10%
	124	5.08%	95.18%
100MHz 4bit	11	4.12%	96.39%
	10	5.17%	97.08
50MHz 4bit	9	3.97%	94.28%

Table 1 This table shows the bin value where a 5% cutoff of electrons was achieved and the resulting rejection of protons at that bin value.

A table was compiled to show how the different ADCs faired against one another. ADC Type is the specification of the ADC that was used. Bin is the bin value on the histogram where a 5% electron cutoff was achieved. And Rejection of Protons shows at that same bin value, how many protons where rejected.

A higher frequency ADC with more bits results in a better resolution and ultimately, better results. Although the 100MHz 4bit ADC rejects more protons than the 100MHz 8bit ADC, it loses electrons in the process which isn't desirable. Therefore, using 100MHz 8bit ADCs to digitize the signals would be preferable due to a higher amount of electron signals saved and a greater amount of proton signals rejected. This shows that you should expected to see that of all the signals that are passed into the ADC, at 15mV, you expect to have around 3% of those signals to be electrons.

Conclusion

Based on the research conducted for the time division simulations, it can be said that applying a low-pass filter can significantly reduce the effects of noise. A 200MHz low-pass filter appears to be a better choice to make when filtering these signals. The increased frequency results in outputting higher current signals than what the 100MHz can output. And there is a slight advantage using the 200MHz low-pass filter when measuring the Δt of the two signals. Even after filtering, noise can still prove to be a hazard when attempting to regenerating the trajectory of the particles in the Tracker.

The ADC Case Study shows expected results; higher performing ADCs will provide a better resolution. An 8bit ADC at 100Mhz rejects ~97% of proton signals while keeping ~95% of the electron signals. Lower frequency ADCs and 4bit ADCs show no useful applications based on the results of the 100Mhz 8bit ADC.

Acknowledgements

I would like to take the SIST committee for accepting me into this great program and providing me an opportunity to see science at work and work with some of the best minds in the world.

I would like to thank my supervisor, Vadim Rusu, and Asset Mukherjee, for helping me along the way with my research.

I would also like to thank Dr. Davenport for assisting me with developing my paper.

And a thanks to my fellow SIST interns for making this a great summer.

References

[1] Blum, W. Rolandi, L. *Detection with Drift Chambers*. 2nd. . Berlin, Germany:Springer-verlag. 1993, 124-125, 168-169. Print.

[2]   "Documentation." *ROOT | A Data Analysis Framework*. July 13, 2012. CERN, Web. . <http://root.cern.ch/drupal/content/documentation>.

[3]   Fermi national accelerator laboratory. United states department of energy. *Mu2e Conceptual Design Report*, Batavia, IL: Fermi Research Alliance, 2012.

[4]   Nenzi, Paolo. "Ngspice documentation." *Ngspice circuit simulation*. July 30, 2012. Nenzi paolo, Web. . <http://ngspice.sourceforge.net/docs.html>.

[5]   Veenhof, Rob . "Http://garfield. Web. Cern. Ch/garfield/." *Garfield - Simulation of Gaseous Detectors*. 7 Sep 2010. CERN, Web. . <http://garfield.web.cern.ch/garfield/>.

Appendix 1-A

Garfield Source file used in the study

```
*******************************************************
* This script plots the arrival time distribution     *
* for the first electron from a 100MeV electron       *
* track through the drift tube.                  *
*                                           *
* NOTE: Garfield help pages can be found at         *
*   http://consult.cern.ch/writeup/garfield/help/    *
*******************************************************


Global threshold=-0.2

***************** CELL ******************************
******************************************************
&CELL
Global xoff=0.0000
Global yoff=0.0000
Global rWire = 0.00125
Global Bx = 0.


** Create a tub with r=.25cm and V=0V
Tube r 0.25 v 0
** Create a wire with the rows command
** syntax:
**  Rows
**  label n diameter x y [V [weight [length [density]]]]
**  (blank line)
Rows
  s 1 2*{rWire} {xoff} {yoff} 1400

****************** MAGNETIC *************************
******************************************************
&Mag
** set the components of the magnetic field.
comp {Bx} 0 0 T

**************** GAS *********************************
******************************************************
&GAS
Global temp=295
Global p = 1
** set the pressure.
pressure {p} bar
Global gas_file=`Ar80-CO220--B{Bx}T--P{p}bar.gas`
Global gas_member `exb`
** if the gas file exists, use it because a call to
**  Magboltz can take a long time.
Call inquire_member(gas_file,gas_member,`gas`,exist)
```

```
If exist then
   get {gas_file,gas_member}
Else
  ** Invoke Magboltz and save the data.
  write {gas_file,gas_member}
  Magboltz argon 80 co2 20 e-field-range 100 500000 ...
    n-e 15 coll 10 mobility 1
Endif

** mobility taken from cern-thesis-98-021
** mobility is in units of cm2/Vs and ep in units of V/cm torr


Read-vector ep mobility
<mobility.mob

** transform to V/cm2 musec which is the garfield unit
Global mobility=mobility*1e-6
add ion-mobility mobility vs ep


**  Call HEED for simulation of ionization of a particle
**  transversing through the gas
Heed argon 80 co2 20

******************** OPTIMISE ************************
******************************************************
&Opt
** set the penning transfer rate of all excited Ar
**  to 30%
penning-transf Ar* 30



&DRIFT
Int-par int-acc 1e-10 mc-dist 0.002 projected-path compute-if-interpolation-
fails
**Int-par int-acc 1e-10


Global ydrift=0.11
Global xdrift=SQRT(0.25*0.25-ydrift*ydrift)

track -{xdrift} {ydrift} 0 {xdrift} {ydrift} 0 electron energy 100 MeV
track heed
clustering-histograms iterations 1000


&SIGNAL


  For i From 1 To 1 Do

window 0.0 0.0001 2000
```

```
Global ypos=0.25*rnd_uniform
Global xpos = SQRT(0.25*0.25-ypos*ypos)

say 'Position={xpos} {ypos}'


**track -{xpos} {ypos} 0 {xpos} {ypos} 0 proton energy 10 MeV ...
**        delta-electrons notrace-delta-electrons

track POSITIONHOLDER ...
        delta-electrons notrace-delta-electrons

**track -{xpos} {ypos} 0 {xpos} {ypos} 0 gamma energy 6 KeV ...
**         delta-electrons notrace-delta-electrons


  track heed
  aval FIXED 30000
**aval townsend

  prepare-track


  signal avalanche  noattachment diffusion ion-tail
            average-signal 2 new

** convolute-signals range 0 1000 transfer-function (5*t/0.01)^5*exp(-
5*t/0.01)
**  plot-signals
write-signals  file=TMPFILE units microampere microsecond

enddo

&STOP
```

End of Garfield source file

Appendix 1-B

Ngspice source file: simulates the pre-amp circuit

```
****************************************************************
* source PREAMP5
*
* Preamp for straws
*
* Preamp circuit by: Vadim Rusu
*
* Tossed together for ngspice by: Daniel Kulas
*
****************************************************************
*   Infineon    Technologies    AG
*   GUMMEL-POON MODEL   IN  SPICE   2G6 SYNTAX
*   VALID   UP  TO  10  GHZ
*   >>> BFP720ESD   <<<
*   (C) 2010    Infineon    Technologies    AG
*   Version 1.0 Juni    2010
****************************************************************

.OPTION TNOM=25, GMIN= 1.00e-12
*BFP720ESD C B E
*$
.SUBCKT BFP720ESD 1 2 3
CBEPAR 22 33 1.048E-013
CBCPAR 22 11 2.58E-014
CCEPAR 11 33 2.737E-013
LB    22 20 6.327E-010
LE   33 30 1.864E-010
LC   11 10  5.957E-010
CBEPCK 20 30  9.242E-014
CBCPCK 20 10  1.779E-015
CCEPCK 10 30  8.276E-014
LBX    20 2 3.338E-010
LEX   30 3 9.323E-011
LCX   10 1  2.42E-010
R_Tr 44 4 683.3
D1 33 25 M_D1
D2 4 25  M_D2
RBLfdb 22 25 1.828
RPS 33 4 0.1123
RSUB 30 4 0.05469
D3 4 15 M_D3
D4 23 33 M_D4
D5 23 15 M_D5
RLDNBL 15 11 6.471
Q1 11 22 33 44 M_BFP720ESD
.MODEL M_D1 D(
+ IS=2.5E-017
+ N=1.02
+ RS=6.1
+ CJO=1.968E-014)
.MODEL M_D2 D(
+ IS=2E-018
+ N=1.02
+ RS=4170
+ CJO=4.284E-015)
.MODEL M_D3 D(
+ IS=3.5E-015
+ N=1.02
+ RS=1380
+ CJO =9.378E-014)
.MODEL M_D4 D(
+ IS=3.5E-015
+ N=1.02
+ RS=0.2
+ CJO =3.128E-014)
.MODEL M_D5 D(
+ IS=3.5E-015
+ N=1.02
+ RS=4.7
+ CJO =5.321E-014)
.MODEL  M_BFP720ESD NPN(
+   IS  =   7.612E-016
+   BF  =   518.4
+   NF  =   1.026
+   VAF =   157.5
+   IKF =   0.05529
+   ISE =   5.344E-015
+   NE  =   1.829
+   BR  =   264.6
+   NR  =   0.9669
+   VAR =   2.278
+   IKR =   0.002409
+   ISC =   4.758E-015
+   NC  =   1.568
+   RB  =   8.442
+   IRB =   0
+   RBM =   0.1186
+   RE  =   0.05132
+   RC  =   2.182
+   XTB =   -2.1
+   EG  =   1.11
+   XTI =   0.1
+   CJE =   5.895E-014
+   VJE =   1
+   MJE =   0.9539
+   TF  =   2.521E-012
+   XTF =   17.49
+   VTF =   0.5295
+   ITF =   0.5638
+   PTF =   4.667
+   CJC =   8.027E-014
+   VJC =   0.4174
+   MJC =   0.3969
+   XCJC    =   0.4894
+   TR  =   1.793E-009
+   CJS =   5.433E-014
+   MJS =   0.6481
+   VJS =   0.6332
+   FC  =   0.7712
+   KF  =   1.264E-010
+   AF  =   1.672)
****************************************************************
.ENDS BFP720ESD
*
*
****************************************************************
* BFR181 NPN
***********************
.SUBCKT BFR181 200 100 300
L1    1   10    0.85nH
L2    2   20    0.001nH
L3    3   30    0.69nH
C1   10   20    84fF
C2   20   30    165fF
C3   30   10    73fF
L4   10  100    0.51nH
L5   20  200    0.49nH
L6   30  300    0.61nH
Q1   2 1 3 BFR181
.ENDS
.MODEL BFR181 NPN(
+ IS = 1.0519e-18     BF = 96.461       NF = 0.90617
+ VAF = 22.403        IKF = 0.12146     ISE = 1.2603e-14
+ NE = 1.7631         BR = 16.504       NR = 0.87757
+ VAR = 5.1127        IKR = 0.24951     ISC = 1.1195e-17
+ NC = 1.6528         RB = 9.9037       IRB = 0.00069278
+ RBM = 6.6315        RE = 2.1372       RC = 2.2171
+ CJE = 1.8168e-15    VJE = 0.73155     MJE = 0.43619
+ TF = 1.7028e-11     XTF = 0.33814     VTF = 0.12571
+ ITF = 0.0010549     PTF = 0           CJC = 3.1969e-13
+ VJC = 1.1633        MJC = 0.30013     XCJC = 0.082903
+ TR = 2.7449e-09     CJS= 0            VJS = 0.75
+ MJS = 0             XTB = 0           EG = 1.11
+ XTI = 3             FC = 0.99768)

****************************************************************
*
*
* PREAMP CIRCUIT
*
*
* ...maybe one day, I'll change the node names to
* something a bit more readable...
*
*
*
```

```
****************************************************************
*
*
*
*
*
*
AV1 %i([N1323970 0]) filesrc
.model filesrc filesource (file="TEMPFILE" amploffset=[0 0] amplscale=[1 1]
+                          timeoffset=0 timescale=1
+                          timerelative=false amplstep=false)
*
*
R_R87         0 N1323970  110k
C_C30         N1004939 N1004929  1p
R_RoutA5      N1195685 N1004949  50
V_V8          N1005055 0 2.5Vdc
R_R64         N1004939 N1004929  5.6k
C_C24         0 N1004987  100n
C_C25         0 N1005233  100n
R_R65         N1004987 N1004939  1k
C_C36         0 N1004929  5.1p
X_Q30         N1005075 N1005075 0 BFR181
X_Q32         N1004949 N1005233 N1004899 BFR181
R_RoutB5      N1195685 N1004933  50
R_R46         N1005075 N1005055  500
X_Q33         N1004933 N1004939 N1004899 BFR181
C_C35         0 N1312121  3p
X_Q31         N1004899 N1005075 0 BFR181
R_R49         N1004929 N1005055  330
R_R66         N1005233 N1004929  5.6k
R_R47         N1005587 N1004929  5.6k
R_R62         N1202523 N1312121  20
X_Q29         N1004929 N1005587 0 BFP720ESD
R_R80         0 N1005587  6.8k
R_R63         N1312121 N1005587  20
V_V9          N1195685 0 2.5Vdc
C_Cin5        N1323970 N1202523  1n
*
*
.tran 0.1ns 0.1us
.control
set filetype=ascii
run
write TEMPFILE N1004933-N1004949
*plot N1004933
.endc
*
*
.END
```

End ngspice file

Appendix 1-C

Functions used for Time Division analysis

```
#ifndef __PLOTSIGNALS_FUNCTIONS
#define __PLOTSIGNALS_FUNCTIONS

/* plotsignals_functions.h

  -Header provides various functions that can be used to manipulate your
input data
  Created By: Daniel Kulas

*/

#include <iostream>
#include <math.h>
#include "TH1D.h"
#include "TGraph.h"
#include "TCanvas.h"
#include "TRandom3.h"
#include "TObject.h"
#include "TLine.h"
#include "TMath.h"
#include "TH1.h"

#define PI  3.1415926535
#define TEMPERATURE 300
#define RESISTANCE  300

//creates canvas and histograms
TCanvas *c1 = new TCanvas("c1","Plot highpass",1200,800);
TH1D    *h1 = new TH1D("h1","Maximum current: Gammas on 100MHz lowpass
",100,-20,0);
TH1D    *h2 = new TH1D("h2","Time distribution: Gammas on 100MHz
lowpass",100,-0.01,0.01);

TGraph* grhp1;
TGraph* grhp2;
TGraph* grhp3;
TGraph* grhp4;

TLine *baseLine  = new TLine(0,0,0.25,0);             //shows were 0 is
relative to signal
TLine *thresLine;
TLine *thresLine2;
TLine *adcHitLine;

TRandom3 *ran   = new TRandom3();
TRandom  *noise = new TRandom();

Double_t newThreshold_sig1, newThreshold_sig2, newThreshold_sig3, fixedThres;
Double_t maxsig = 0;
Double_t adcHit;

float noiseCurr = 0;

// threshold: -0.11 @ 30MHz  -0.21 @ 100MHz, -0.26 @ 200MHz, <----Used only
if dynamic threshold is taken out
// threshold values are based on noise on a lowpass at set freqs

bool highFilter, lowFilter;
const char* theFile;



/*==================Filters========================
  ==================================================*/
//Applys a highpass filter to the input signal
void highpass(Double_t* x, int n, Double_t dt, Double_t frequency, Double_t
R, Double_t* y)
{
    Double_t RC = 1/(2 * PI * frequency);
    Double_t alpha = (RC / (RC + dt));
    y[0] = 0;
    for (int i = 1 ; i < n; i++)
    {
        y[i] = alpha * y[i-1] + alpha * (x[i] - x[i-1]) + R * x[i];
    }
};

//Applys a lowpass filter to the input signal
void lowpass(Double_t* x, int n, Double_t dt, Double_t frequency, Double_t*
y)
{
    Double_t RC = 1/(2 * PI * frequency);
    Double_t alpha = dt / (RC + dt);
    y[0] = 0;
    for(int i = 1; i < n; i++)
    {
        y[i] = alpha * x[i] + (1 - alpha) * y[i-1];
    }
};

Double_t adcdigi(int n, Double_t* x, Double_t* y)
{
    Double_t maxsignal = 0;
    Double_t freq = 30;   //MHz
    Double_t ranVar = ran->Rndm();
    Double_t dt = x[n-2] - x[n-3];
    Int_t adcfreq = 1/(freq * dt);                     //adc frequency
    Int_t startdigi = adcfreq * ranVar;                //conversion starts at a
random point

    for (int i = 0 ; i < n ; i++)
    {
        Int_t startConversion = i-startdigi;         //starting point of
conversion at random time
        if ( startConversion % adcfreq == 0 )
        {
            if (y[i] < maxsignal)
            {
                maxsignal = y[i];
                adcHit = x[i];
            }
        }
    }
    return maxsignal;
};

Double_t findMax(int n, Double_t* y)
{
    Double_t maxsignal = 0;
    for(int i = 0; i < n; i++)
    {
        if(y[i] < maxsignal)
        {
            maxsignal = y[i];
        }
    }
    if(maxsignal > getFixedThres())
        maxsignal = 1;
    return maxsignal;
};

Double_t getadcHit()
{
    return adcHit;
};


/*==================Threshold======================
  ===================delta t=======================*/

//fix logic to account for different noise levels
void setUserFixedThres(int userVal)
{
    if(userVal == 30)
        fixedThres = -0.11;
    else if(userVal == 100)
        fixedThres = -0.21;
    else if(userVal == 200)
        fixedThres = -0.26;
    else
        fixedThres = -0.21;
}

//dumb code..forgot what it does but I'm sure has some purpose
void setFixedThres()
```

```
{
    fixedThres = fixedThres;
}

Double_t getFixedThres()
{
    return fixedThres;
}

void setMaxVar()
{
    maxsig = 0;
}

void setNewThres_sig1(int n, Double_t* y)
{
    newThreshold_sig1 = 0;
    maxsig = 0;

    for(int i = 0; i < n; i++)
    {
        if(y[i] < maxsig)
        {
            maxsig = y[i];
        }
    }

    if(maxsig >= getFixedThres())
    {
        cout << "Don't change the threshold. Signal 1 does not cross fixed
threshold" << endl;
    }
    if(maxsig <= getFixedThres())
    {
        cout << "Signal 1 crosses fixed threshold" << endl;
        newThreshold_sig1 = maxsig/2;
        if(newThreshold_sig1 >= getFixedThres())
        {
            cout << "New threshold is lower than fixed threshold. Going back
to fixed threshold" << endl;
            //cout << "New threshold at: " << newThreshold_sig1 << endl;
            newThreshold_sig1 = fixedThres;
        }
        cout << "Threshold set at: " << newThreshold_sig1 << " uA on signal
1" << endl;
    }
     cout << "" << endl;
}

void setNewThres_sig2(int n, Double_t* y)
{
    newThreshold_sig2 = 0;
    maxsig = 0;
    for(int i = 0; i < n; i++)
    {
        if(y[i] < maxsig)
        {
            maxsig = y[i];
        }
    }

    if(maxsig >= getFixedThres())
    {
        cout << "Don't change the threshold. Signal 2 does not cross fixed
threshold" << endl;
    }
    if(maxsig <= getFixedThres())
    {
        cout << "Signal 2 crosses fixed threshold" << endl;
        newThreshold_sig2 = maxsig/2;
        if(newThreshold_sig2 >= getFixedThres())
        {
            cout << "New threshold is lower than fixed threshold. Going back
to fixed threshold" << endl;
            //cout << "New threshold at: " << newThreshold_sig2 << endl;
            newThreshold_sig2 = fixedThres;
        }
         cout << "Threshold set at: " << newThreshold_sig2 << " uA on signal
2" << endl;
    }
     cout << "" << endl;
}

void setNewThres_sig3(int n, Double_t* y)
{
    newThreshold_sig3 = 0;
    maxsig = 0;

    for(int i = 0; i < n; i++)
    {
        if(y[i] < maxsig)
        {
            maxsig = y[i];
        }
    }

    if(maxsig >= getFixedThres())
    {
         cout << "Don't change the threshold. Signal 3 does not cross fixed
threshold" << endl;
    }
    if(maxsig <= getFixedThres())
    {
        cout << "Signal 3 crosses fixed threshold" << endl;
        newThreshold_sig3 = maxsig/2;
        if(newThreshold_sig3 >= getFixedThres())
        {
            //cout << "New threshold at: " << newThreshold_sig3 << endl;
            cout << "New threshold is lower than fixed threshold. Going back
to fixed threshold" << endl;
            newThreshold_sig3 = fixedThres;
        }
          cout << "Threshold set at: " << newThreshold_sig3 << " uA on signal
3 [no noise]" << endl;
    }
     cout << "" << endl;
}

Double_t getNewThreshold_sig1()
{
    return newThreshold_sig1;
}

Double_t getNewThreshold_sig2()
{
    return newThreshold_sig2;
}

Double_t getNewThreshold_sig3()
{
    return newThreshold_sig3;
}

//cycle through signal. Checks each value to see if it goes over threshold
bool did_sig_miss_thres(int n, Double_t* y)
{
    Double_t maxVal = 0;

    for(int i = 0; i < n; i++)
    {
        if(y[i] <= maxVal)
        {
            maxVal = y[i];
        }
    }
    if(maxVal < getFixedThres())
    {
        //   cout << "Signal did not miss threshold" << endl;
        return false;
    }

    return true;
};

//calcTime1 & calcTime2 return the time when the signal crosses the threshold
for the first time
//Only calculates time once per signal (assume if crosses threshold)
Double_t calcTime1(int n, Double_t* x, Double_t* y)
{
    long double time_1;
    for (int i = 0 ; i < n ; i++)
    {
        if( y[i] < getFixedThres() )
        {

            if ( y[i] < getNewThreshold_sig1())     //if signal is greater
than threshold, calc delta t
            {
                time_1 = x[i];
                   cout << "Threshold for sig 1: " << getNewThreshold_sig1()
<< " At time = " << x[i] << " us" << endl;
                return time_1;
            }
        }
    }
    return time_1;
};

Double_t calcTime2(int n, Double_t* x, Double_t* y)
{
    long double time_2;
    for (int i = 0 ; i < n ; i++)
    {
        if( y[i] < getFixedThres() )
        {
            if ( y[i] < getNewThreshold_sig2())
            {
                time_2 = x[i];
                  cout << "Threshold for sig 2: " << getNewThreshold_sig2()
<< " At time = " << x[i] << " us" << endl;
                   cout << "" << endl;
                return time_2;
            }
        }
    }
    return time_2;
};

Double_t calcRMS(Double_t n, Double_t* y)
{
    Double_t sumY = 0, rms;

    for(int i = 0; i < n; i++)
    {
        sumY += pow(y[i], 2);
    }
    return rms = sqrt(sumY/n);
};

void setNoiseCurr(int userBand)
{
    noiseCurr = sqrt((4*(TMath::K())*TEMPERATURE*userBand*1E6)*300)/300;
    cout << "The current noise is set at " << noiseCurr*1E6 << " uA." <<
endl;
};
```

```
Double_t getNoiseCurr()
{
    return noiseCurr*1E6;
};

void DEBUG()
{
    cout << "TESTING" << endl;
};
#endif
```

Appendix 1-D

Script used to generate data for the ADC case study

```
#!/bin/bash
################################################################################
#########################
#   Script used to read in the data files, format it, and pass it as
parameters to runprocesses.sh
#   Those parameters are used to modify the garfield .in file
#
#   $filecounter tells the script where it is at in the data file so when we
create the output file
#   from spice, it will give it a name like "signal_[POSITION IN DATA
FILE].dat"
#
#   $count is used to keep track of how many processes are currently running.
#   If count is equal to the amount of processes running, wait for those
processes to finish.
#
#   User decides how many different processes to run at once.
#   Thing to note: 1 process takes up about 160Mb of RAM
#                  8 processes takes up about 1.24GB of RAM
#                  [x] processes * 160 = memory usage
#
#   Daniel Kulas
#   7/30/12
#   mu2e
################################################################################
#########################

clear

#the data files
ELEC=electrons
PROT=protons

STORAGE=storage/
GARFSTORAGE=garfsrc_storage/
GARFDATSTORAGE=garfdat_storage/
rm -rf $GARFSTORAGE
mkdir $GARFSTORAGE
rm -rf $GARFDATSTORAGE
mkdir $GARFDATSTORAGE
rm -rf $STORAGE
mkdir $STORAGE

#template file for ModGarf
TEMPLATE_DIR=templates/
TEMPLATE_GARF_ELEC=$TEMPLATE_DIR"garfield_electron.in"
TEMPLATE_GARF_PROT=$TEMPLATE_DIR"garfield_proton.in"
```

```
echo $TEMPLATE_GARF_PROT

if [ ! -f $TEMPLATE_GARF_PROT ]; then
    echo "No proton template for garfield";
fi
if [ ! -f $TEMPLATE_GARF_ELEC ]; then
    echo "No electron template for garfield";
fi

count=0
filecounter=1

echo "Enter amount of processes to run at a time"
read procs
[[ $procs = *[![:digit:]]* ]] && echo "Not an Integer. Try again" && exit

echo "Simulate electrons or protons?"
echo "----E)lectrons----P)rotons----"
read selection_sig

if [ $selection_sig == "E" -o $selection_sig == "e" ]; then

    #formats file
    sed -i 's/\,/ /g' $ELEC
    sed -i 's/(//g' $ELEC
    sed -i 's/)//g' $ELEC
    sed -i 's/\t/ /g' $ELEC

    #reads number of lines in file
    numberoflines=`wc -l $ELEC| awk '{print $1}'`
    if [ $numberoflines -gt 100000 ]; then echo "Error?"
        exit
    fi

    #reads line by line of the file
    cat $ELEC | while read line; do

        #runs multiple processes of the runprocesses.sh script
        ./runprocesses.sh $line $filecounter $TEMPLATE_GARF_ELEC &

        #keep track of how many processes are running
        count=$[$count+1]
        #keep track of what line you are on in the file
        filecounter=$[$filecounter+1]

        #if you count is equal to the number of processes currently running,
wait
        if [ $count -eq $procs ]; then
            wait
            count=0
        fi
        #wait until all lines of the data file are read through
        if [ $filecounter -eq $numberoflines ]; then
            wait
        fi

    done

    echo "Combining outputs to newoutput.dat"

    #remove any old .dat file left over from the previous run and combine the
outputs generated
    #by runprocesses.sh
    rm newoutput_electrons.dat
    cat storage/signal* > newoutput_electron.dat

elif [ $selection_sig == "P" -o $selection_sig == "p" ]; then
    #formats file
    sed -i 's/\,/ /g' $PROT
    sed -i 's/(//g' $PROT
    sed -i 's/)//g' $PROT
    sed -i 's/\t/ /g' $PROT
    numberoflines=`wc -l $PROT| awk '{print $1}'`

    echo "The file has "$numberoflines" lines"
    if [ $numberoflines -gt 100000 ]; then echo "Error?"
        exit
    fi
    cat $PROT | while read line; do

        filecounter=$[$filecounter+1]
        ./runprocesses.sh $line $filecounter $TEMPLATE_GARF_PROT&

        count=$[$count+1]

        if [ $count -eq $procs ]; then
            wait
            count=0
        fi

        if [ $filecounter -eq $numberoflines ]; then
            wait
        fi

    done

    echo "Combining outputs to newoutput.dat"

    rm newoutput_protons.dat
    cat storage/signal* > newoutput.dat

else
    echo "Your signal selection input was incorrect. Try again."
fi
exit
```

Appendix 1-D (cont…)

runprocesses.sh script: Runs the programs to generate data

```
#!/bin/bash
#   Daniel Kulas
#   7/28/12
#
# Generates data in a unique folder based on the processes PID, moves the
useful data out, and removes all
# unnecessary data from the current folder.
#
# $1 = energy
# $2 = xpos0
# $3 = ypos0
# $4 = zpos0
# $5 = xpos1
# $6 = ypos1
# $7 = zpos1
# $8 = The current line of the data set
# $9 = The template garfield source file
#
# $$ gets the pid of the script     $! gets the pid of the background process
PIDFILE=$$
FILE="file_"$PIDFILE
mkdir $FILE
FILE_DIR=$FILE

STORAGE=storage/
GARFSTORAGE=garfsrc_storage/
GARFDATSTORAGE=garfdat_storage/

TEMPLATE_GARF=$9

#the resulting file generated from ModGarf. This file is already insie of
$FILE_DIR
TMP_GARF_FILE="tmpgarf.in"
NEW_GARF_FILE="garf_"$8".in"

#the data generated from garfield
TMP_GARF_OUT="TMPgarfile"
NEW_GARF_OUT="garfFile_"$8".dat"

#the file generated from formatting
FORMATTING_OUT="reduced_signal"

#Garfield2Spice output
GAR2SPICE_OUT="formatted_signal.dat"

#the spice template file
TEMPLATE_SPICE="PreampTEMPLATE.net"
TMP_SPICE_FILE="Preamp.net"
SPICE_OUT=$(printf 'signal%05d.dat' $8)
```

```
#Modify the garfield source file to take in the parameters from the data file
and run garfield
./ModGarf_prot $1 $2 $3 $4 $5 $6 $7 $TEMPLATE_GARF $FILE_DIR/$TMP_GARF_FILE
$FILE_DIR/$TMP_GARF_OUT >/dev/null 2>&1
 garfield-9 < $FILE_DIR/$TMP_GARF_FILE >/dev/null 2>&1

#   PRODUCES AN OUTPUT FILE CALLED
echo "Formatting files for SPICE"
./FormatFiles $FILE_DIR/$TMP_GARF_OUT $FILE_DIR/$FORMATTING_OUT


if [ ! -s $FILE_DIR/$FORMATTING_OUT ]; then
    echo "No signals produced. Aborting..."
    exit -1
fi

echo ""
echo "Your file generated by garfield has useful data...continuing..."

./Garfield2Spice $FILE_DIR/$FORMATTING_OUT $FILE_DIR/$GAR2SPICE_OUT
echo ""
echo "Modifing the spice file and running spice..."

#   MODIFY SPICE FILE
./ModSpice $TEMPLATE_SPICE $FILE_DIR/$TMP_SPICE_FILE $FILE_DIR/$GAR2SPICE_OUT
$FILE_DIR/$SPICE_OUT

# -b runs in batch mode, gets data and leaves the process
ngspice -b $FILE_DIR/$TMP_SPICE_FILE    >/dev/null 2>&1

echo ""
echo "Moving "$SPICE_OUT" to "$STORAGE
echo "Deleting the contents of "$FILE_DIR

# move the output file from spice to the storage folder for processing
mv $FILE_DIR/$SPICE_OUT $STORAGE

mv $FILE_DIR/$TMP_GARF_FILE $FILE_DIR/$NEW_GARF_FILE
mv $FILE_DIR/$NEW_GARF_FILE $GARFSTORAGE

mv $FILE_DIR/$TMP_GARF_OUT $FILE_DIR/$NEW_GARF_OUT
mv $FILE_DIR/$NEW_GARF_OUT $GARFDATSTORAGE

rm -rf $FILE_DIR

echo "^^^^^^^^^^^^"
echo "Success!"
echo "^^^^^^^^^^^^"
exit 0
bash
```

End runprocesses.sh script