M9 SiPM Spectrometer Work Report

Jerin Roberts
January 13, 2015

Supervisor: Dr. Syd Kreitzman
Locations: TRIUMF

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This report is a summary of the work completed on the M9 prototype spectrometer project during the fall work period (September to December 2014). The summary of my work includes characterizing SiPM devices for front end electronics work, as well as designing/running simulations for timing statistics. Descriptions of the component modifications, simulation scripts, and methods for data collection are described in this report.

2. MUON SPECTROMETRY

2.1. OVERVIEW

Muons are an elementary particle (lepton) similar to the electron, with negative electric charge (-1) and a spin of 1⁄2 and a mass of (105.7 MeV/c2). The muon is an unstable subatomic particle with a mean lifetime of 2.2 μs. The decay of the muon is mediated by the weak interaction exclusively. Muon decay always produces at least three particles, which must include an electron of the same charge as the muon and two neutrinos of different types as displayed in figure 2.1. For positively charged muons a positron and two neutrinos are released during decay. The trajectory of the released positron/electron is well known relative to the spin axis of the decaying muon. This artefact of the decay provides a key mechanism for studying internal magnetic structures of materials.

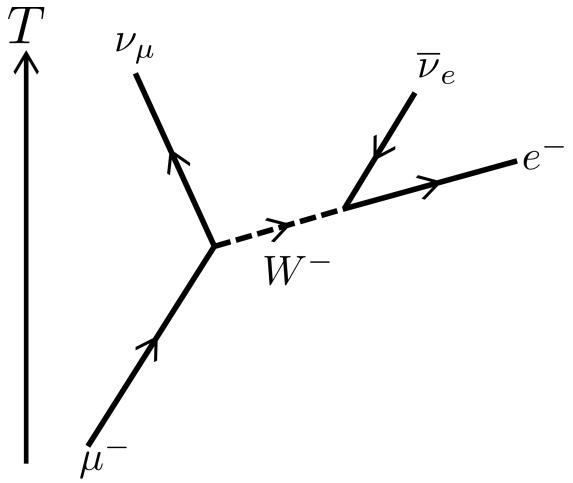


Figure 2.1: Displays a diagram of a muon and its decay constituents. Due to charge conservation the electron charge will match the charge of the decaying muon.

2.2. PRESENT SPECTROMETERS

Muon Spectrometers are designed to probe the fine magnetic structures of materials being studied. A simplified diagram of a muon spectrometer is shown in figure 2.2. Since muons have a short lifetime (2.2us), the spectrometers must have a ready supply of freshly generated

muons. At TRIUMF protons are accelerated using the institutions large cyclotron. These are then fired at a boron target which releases positively charged muons. These are collected and focused into and down a beam line leading to the spectrometer. While traveling down the beam line the spin axis of the muon is pointed opposite of the particles momentum vector or more specifically, the beam is spin polarized. The muons are guided into the material being studied and since they are at a lower energy they are stopped and embedded in the material. While the muons are in the material the spin of the muon precesses about the local internal magnetic field of the sample. When the muons finally decay the positron is released in the direction of the precessing spin of the muon. The target in this spectrometer resides at the center of the device and is surrounded by scintillating material. When the positron passes through the scintillator piece a burst of light is generated. The light travels down to the ends of the scintillator, and in some spectrometers may pass through into a wavelength shifter light guide. PMT's are positioned at the end of the light guides to detect the incoming light. Each signal is given a time stamp where the timing or time difference between PMT's at either ends of the device gives the point where the positron passed through the scintillation piece. This allows for the positron trajectory to be reconstructed which ultimately gives information about the shape of the internal magnetic field of the sample. Also displayed in the diagram is muon counters which help discriminated against noise through coincidence and to further help predict positron trajectory. All current muon spectrometers employ the use of photomultiplier

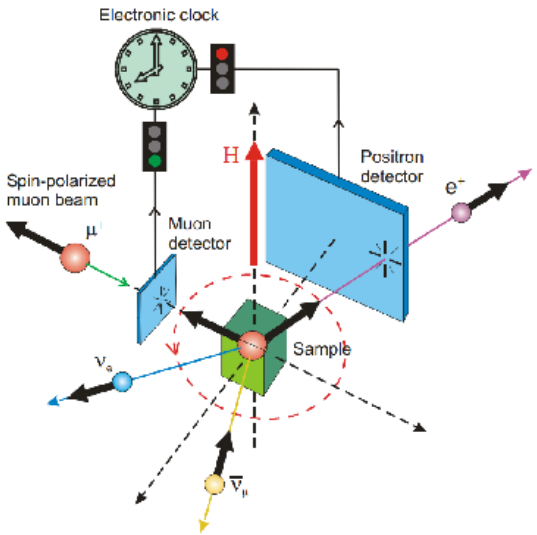


Figure 2.2: Displays a simplified diagram of a conventional muon spectrometer found at TRIUMF.

tubes (PMT's) for light detection. PMT's make use of the photo-electric effect and secondary emission for light detection. A schematic of one is shown in figure 2.3 . A photon incident on the device first collides into a photo-cathode screen where a primary electron is produce at an efficiency of about 1/4 (quantum efficiency). The primary electron is then accelerated through a focusing electrode onto the first dynode. Upon colliding with the surface secondary

emissions occur releasing a burst of electrons. These are then again accelerated to another dynode again releasing more electrons during collision. This amplifying effect continues until the anode is reach for which the electrons are conducted out of the device as a electrical signal. The PMT effectively turns a single photon into millions of electrons enabling low light level detection. Because of the high bias used to power the dynodes the PMT produces a high signal to noise ratio, which makes them optimal for timing applications. However in a muon spectrometer PMT's can have many downsides. In addition to there high cost, PMT's can be cumbersome to work with. For example regular PMT's can not be exposed to a high magnetic

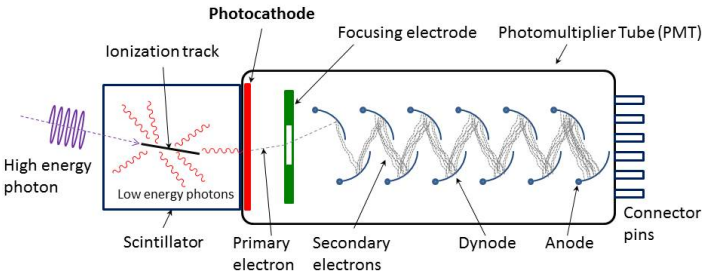


Figure 2.3: Displays a simplified diagram of a conventional photomultiplier tube (PMT).

field, and therefore special shielding is required in order to use them in the spectrometer. However the use of shielding can distort the magnetic field. In order to preserve the field and focused muon beam, placement of the PMT's requires symmetrical positioning and in some cases the addition long wave guides. This again limits spectrometer design and increases costs. These problems could potentially be solved with inexpensive silicon photomultiplier's.

2.3. SiPM

An avalanche photodiode (APD) is a sensitive semiconductor device that exploits the photoelectric effect to convert light to an electric signal. APDs can be thought of as photodetectors that provide a gain through avalanche multiplication and can be considered as the semiconductor analog to photomultipliers. Figure 2.4 displays a simplified diagram of the internal structure of a ADP. Silicon photomultipliers (SiPM's) are Silicon photon sensitive devices built from an array of APD's operating in Geiger mode. During avalanche breakdown when applying an external electric field to a semiconductor holes and electrons move to opposite ends. If the E field is high enough electrons can eject neighboring electrons, creating more holes and free electrons. The fast moving holes allow more e-hole pairs to form which means the crystal becomes conducting. In an Avalanche Diode the original carrier is created by the absorption of

a photon. Once absorbed an electron is eventually ejected and in the presence of an external e-filed is accelerated which frees more electrons and holes. Newly ejected electrons also get accelerated which still have the ability to knock others free. The result is a shower of electrons and holes, resulting in measurable current, started by one photon.

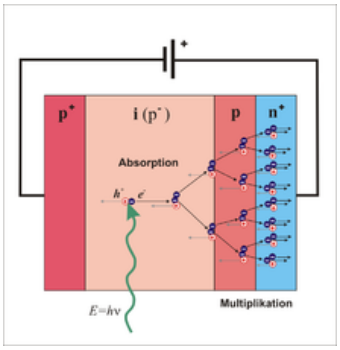


Figure 2.4: Displays a simplified diagram of a conventional avalanche photodiode (APD).

The dimension of each single APD can vary from 20 to 100 micrometres. The supply voltage (Vb) depends on APD technology used, and typically varies between 20 V and 100 V, thus being from 15 to 75 times lower than the voltage required for a traditional photomultiplier tubes (PMTs) operatio. Photo detection efficiency (PDE) ranges from 20-50% depending on device and wavelength. Gain of the device is also similar to a PMT being approximately 10^6. The gain verse over-bias voltage dependence is linear, unlike PMT which follow the power law. SiPM signals are unaffected by external magnetic fields and because of their light, small design have little impact on the field itself. These are the key features which make SiPM's a viable option for future spectrometer design.

3. Simulation

3.1. Introduction

A large portion of work complete on this project involved simulating the effects and geometry and discrimination on timing. A simulation was written from scratch in c++ and analyzed using ROOT. The simulation was designed to incorporate more geometric complex scintillation pieces. The architecture enables simulation of scintillation pieces with no flat surfaces. This portion describes the simulations in detail for future modification ability. For the purpose of clarity each single SiPM device will be referred to as a "pixel" while each pixel on the device will be referred to as "micro-pixels" (Sensl convention).

3.2. Photon Generation

The program first generates random photon paths inside the geometry. It accomplishes this by first randomly selecting a start point which lies along a path simulating a positron/electron passing through scintillator.

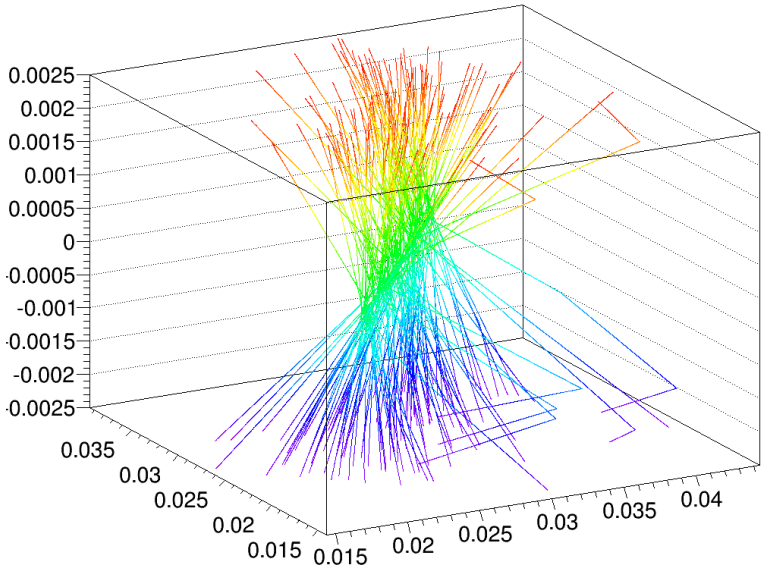


Figure 3.1: Displays simulated photon paths originating from positron path.

The simulation has bee written such that one has the ability to choose a specific path for the positron, as well as defining a boundary in which a positron path will be randomly generated within. This simulates the uncertainty found in the spin precession of the decaying muon. The code contains lines for both a "hard set" path or a range that a path can be random generated in.

```
std::vector<float> scintx (np), scinty (np), scintz (np);
float zomp, jomp, xs, ys, radt, radts;
//tomp = tan(gRandom->Uniform(0.698,0.8726));//slope ranges NO NEGS
 tomp = tan(0.78);
// zomp = gRandom->Uniform(-0.087,0.087);//angle range (rads)
 zomp = 0.0;
```

The lines above (currently in "hard set" mode) show the parameters that can be adjusted to set the path of the incoming positron/electron. The variable 'tomp' sets the range in the x-y plane measured in rads from the x-axis where 'zomp' sets the range in x-z plane and again is measured from the x-axis. To confirm these tracks one can run the photon-path visualization scripts also provided.below shows the lines responsible for generating random photons either along a "hard set" path or one that was randomly generated within the range set. Currently the boundaries mark out a "square cone", future work could include making the boundary a cone represented by a inputted solid angle.

```
cout << "generating random start locations...\n";
for (float w = 0; w < np; ++w)
{
```

```
xs = gRandom->Uniform(0,10);
ys = tomp*xs;
radt = sqrt(pow(xs,2) + pow(ys,2));
radts=((rad1)+(radt*((rad2-rad1)/sqrt(200))));
scintz.at(w) = radts*tan(zomp) + stlength;
jomp = radt/radts;
scintx.at(w) = xs/jomp;
scinty.at(w) = ys/jomp;
bool out = (rad2 <= sqrt(pow(scintx.at(w),2) + pow(scinty.at(w),2)));
bool out2 = (rad1 >= sqrt(pow(scintx.at(w),2) + pow(scinty.at(w),2)));
if(scintz.at(w) >= (length/2))
{scintz.at(w) = 0;
cout << "\n ***ERROR : Solid angle beyond scintillator Length****";
}
if(out || out2)
--w;
}
```

The generation of the random starting point follows a uniform distribution where the probability of generating a photon is uniformly distributed over the entire path of the positron as it passes through the scintillator. Once a starting point has been generated (Cartesian coordinates), a velocity vector(referred to as "direction vector" from here on) is also randomly generated with 1 unit magnitude. Each random component follows a uniform distribution and is stored in a vector as displayed below.

```
   for (float i = 0; i < rand.size(); ++i)
     {tomp = gRandom->Uniform(-10,10);
rand.at(i)=tomp;
     }
```

Inside the main loop the components are then normalized so each increment is one step unit (in this case its about a 10um). The program simply "increments" the position vector using the direction vector and checks after each step if the position vector has reached a boundary. Each time a boundary is encountered the direction vector is altered to simulate a reflection. Simple boundaries can be written to simple adjust the sign of the direction vector. This would however only apply for planes perpendicular to the x,y or z axis. For more complicated structures a boundary is represented by a function (in this case a circle) when the photon encounters the boundary, it forms a flat plane by calculating the derivative at that point and generating normal vector of that plane as displayed in the code below. Note, this piece relies on the fact that the function only exists in the x and y plane. This may need to be modified for a 3D function.

```
fprime = -(pos.at(0))/(sqrt(pow(sqrt(pow(pos.at(0),2)+pow(pos.at(1),2)),2)
-pow(pos.at(0),2)));
```

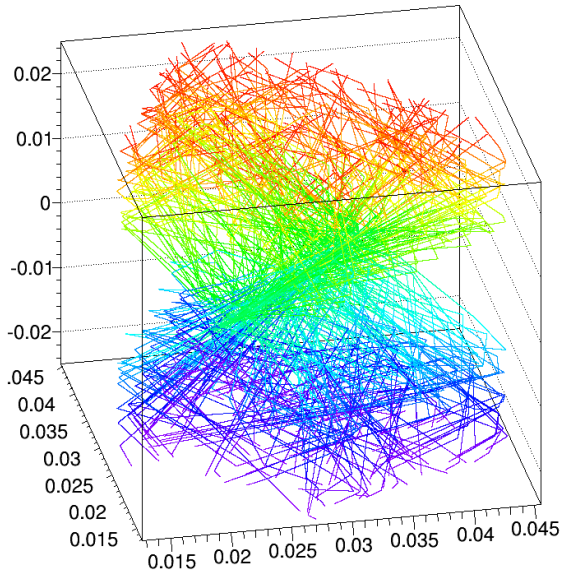
```
//tangent of boundary at intersecting point

//put into vector:
v1.at(0) = 1;
v1.at(1) = fprime;
v1.at(2) = 0;
//second vector parallel to z axis
v2.at(0) = 0;
v2.at(1) = 0;
v2.at(2) = 1;

//cross product of v1 and v2 which gives normal vector of reflecting plane
normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
```

With the normal vector of the plane and the incident vector $\vec{v}_i$ of the photon, the new reflected directional vector $\vec{v}_r$ can be calculated using equation 3.1.

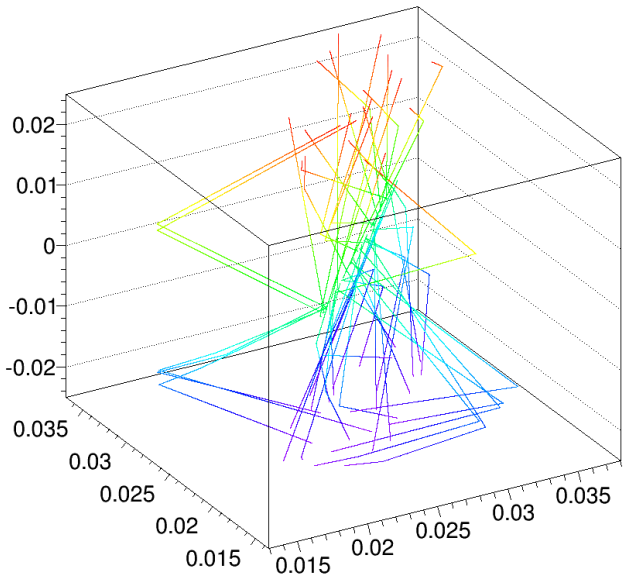
$$\vec{v}_r = \vec{v}_i - (2\vec{n}(\vec{v}_i \bullet \vec{n})) \tag{3.1}$$

The normal vector of the plane, $\vec{n}$ must be normalized. once this new directional vector is calculated, its implemented and added to position vector. At the same time the angle of the incident photon relative to the surface is calculated and checked against the critical angle parameters. If the angle is found to exceed the set parameter, the run variable is set to false, terminating the run without final position or timing logs. If the angle parameters are found to be true the process of incrementing continues until the photon encounters another boundary or one of the end planes. Once this occurs, the time of the path is calculated based on the index of refraction of the scintillator, and the final position is logged which are then both and stored their respected vectors and the process begins for another photon. It should be noted that only photons which have landed inside the pixel array are recorded. Ones outside or not meeting incident angle parameters are discarded as shown below.

```
if(end2)
{
run = false;
xs = gRandom->Uniform(0,100);

//check the angle of incidence
v1.at(0) = 1;
v1.at(1) = 0;
v1.at(2) = 0;

v2.at(0) = 0;
```



(a)



(b)

Figure 3.2: Displays the difference between detected and total photons

```
v2.at(1) = 1;
v2.at(2) = 0;
```

```
normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
//check if crital angle is met
magdir = sqrt(pow(dir.at(0),2)+pow(dir.at(1),2)+pow(dir.at(2),2));
float magnormalv =
 sqrt(pow(normalv.at(0),2)+pow(normalv.at(1),2)+pow(normalv.at(2),2));
float VdotN =
 dir.at(0)*normalv.at(0)+dir.at(1)*normalv.at(1)+dir.at(2)*normalv.at(2);
float theta = acos(VdotN/(magdir*magnormalv));

if(theta <= cang2 && xs <= 25 )
{
if(pos.at(1) <= (pos.at(0) + 0.010) && pos.at(1) >= (pos.at(0) - 0.010))
sensArea2 = true;
}



}
```

If a photon activates a pixel, it has a probability of activating its neighbor simultaneously through phenomenon known as cross-talk. each time a photon hits the sensor its given a "coin toss" on whether or not it will excite its neighbor This probability can be easily adjusted as its unique for each model. If the toss happens to produce a true outcome, another photon is immediately generated one pixel length away from the original, this is then checked to validate its inside the sensor area, and is then again given a prob of exciting its neighbor. the process continues for each newly generated cross talk photon until a "toss" produces a false condition, or the photon lands outside the sensor area. For each cross-talk photon, the path time of the original starting one is recorded, as the program relies on steps to calculate time, the cross-talk photons need to be given the same time as the original exciting photon which is shown below.

```
while(fomp < 20)//defines prob of exciting one neighbor
{
//generates random direction to increment (one pixel size) outside original pixel
float x = gRandom->Uniform(-1,1);
float y = gRandom->Uniform(-1,1);
//normalize for pixel size
float A = sqrt(pow(x/(pixsize),2)+pow(y/(pixsize),2));
x = x/A;
y = y/A;
//apply change to last final position
pos.at(0) = pos.at(0) + x;
pos.at(1) = pos.at(1) + y;
```

```
//check to make sure still on SiPM
bool ta, tb, tc, td;
ta = (pos.at(1) <= (pos.at(0) + 0.010));
tb = (pos.at(1) >= (pos.at(0) - 0.010));
tc = (rad2 >= sqrt(pow(pos.at(1),2)+pow(pos.at(0),2)));
td = (rad1 <= sqrt(pow(pos.at(1),2)+pow(pos.at(0),2)));
if( ta && tb && tc && td)
{
//if true records same pathlength(time) and new final position
if(sensArea){
lpath.push_back (pl);
finalpos.push_back(pos);
++xtalk;
}
if(sensArea2){
lpath2.push_back (pl);
finalpos2.push_back(pos);
++xtalk2;
}

fomp = gRandom->Uniform(0,100);
}
else//stop new excitations if photon leaves SiPM
{fomp = 100;}
   }
```

3.3. DETECTION

The final positions of all photons found in sensory area are also stored and eventually checked for double hits. Once all photon times have been recorded the program adds a scintillation adjustment to the timing. Its known that once a scintillation molecule is excited it may not release its photon immediately, as their is some probability of it jumping down to 1s and then to 0s orbitals. This gives a rise and fall time distribution for each scintillation photon released. The program generates a random timing delay based on a rise and fall distribution function. These are then added to each photon timing calculated earlier as displayed below.

```
for (float i = 0; i < scint.size(); ++i)
     {
tmp = sqroot->GetRandom();
ftime.at(i) = tmp + pathtime.at(i);
     }
```

```
for (float i = 0; i < scint2.size(); ++i)
     {
       tmp = sqroot->GetRandom();
ftime2.at(i) = tmp + pathtime2.at(i);
     }
```

The program then checks for double hits, since each micro-pixel has a dead time once activated. The program searches for photons with similar arrival times and final positions. When two points are found with similar coordinates(within pixel size) and timing(dead time), the one with the largest time is deleted. The insures the ones that first activated the micro-pixel are stored.

```
for(int i = 0; i < 2; ++i)
{

int fpsize = finalpos.size();
int n =0;
bool simhit = true;
while(simhit)
{
temppos = finalpos.at(n);
p = 0;
bool simhit2 = true;
while(simhit2)
{ temppos2 = finalpos.at(p);
bool a, b, c;
a = (abs(temppos2.at(0) - temppos.at(0)) <= pixsize/2);
b = (abs(temppos2.at(1) - temppos.at(1)) <= pixsize/2);
c = (abs(ftime.at(n)-ftime.at(p)) <= deadtime);
if((n!=p) && a && b && c)
{
if(ftime.at(n) > ftime.at(p))
{
ftime.erase (ftime.begin() + n);
finalpos.erase (finalpos.begin() + n);
--n;
++g;
}


simhit2 = false;
}
```

```
if((p+1) == (fpsize -g))
simhit2 = false;
++p;
}
++n;
if(n == (fpsize -g) || (fpsize-g) == 1)
simhit = false;
}
}
```

3.4. FINAL SIGNAL

Next the final signal from each pixel is calculated individually. This is done by summing all the individual micro-pixel signals with the timing adjustment, which is denoted as t_o in equation 3.2.

$$V(t) = V_{max}(exp(\frac{-(t-t_o)}{\tau_{Rise}}) - exp(\frac{-(t-t_o)}{\tau_{Fall}})) \tag{3.2}$$

The program applies each timing adjustment before summing all the signals. The signals are summed and stored in a vector. The size of the vector denotes the resolution of the signal, ie each component of the vector represents a point in the final function. Initially the entire vector is set to zero. Then the timing adjustment is imputed into a function which is then evaluated for each component in the vector. The process continues for each timing adjustment as shown below.

```
for(int n = 0; n < ftime.size(); ++n)
{
for(int i = 0; i < graphs1A.size(); ++i)
{
tomp = -(11.94)*(exp(-((i*.001)-ftime.at(n))/2.341)-exp(-((i*.001)-ftime.at(n))/9.526));
if (tomp > 0)
{
if(Sfinpos1.at(n) == 1)
graphs1A.at(i) = tomp + graphs1A.at(i);
if(Sfinpos1.at(n) == 2)
graphs1B.at(i) = tomp + graphs1B.at(i);
if(Sfinpos1.at(n) == 3)
graphs1C.at(i) = tomp + graphs1C.at(i);
if(Sfinpos1.at(n) == 4)
graphs1D.at(i) = tomp + graphs1D.at(i);
}
else
{
if(Sfinpos1.at(n) == 1)
```

```
graphs1A.at(i) = 0 + graphs1A.at(i);
if(Sfinpos1.at(n) == 2)
graphs1B.at(i) = 0 + graphs1B.at(i);
if(Sfinpos1.at(n) == 3)
graphs1C.at(i) = 0 + graphs1C.at(i);
if(Sfinpos1.at(n) == 4)
graphs1D.at(i) = 0 + graphs1D.at(i);

}
}
}
```

This makes for an efficient and memory saving method for calculating the final function. There are a couple of lines which organize all timings into specific groups corresponding to different pixels. These final signals are then shifted based on a threshold, then the intercept is calculated. This gives the timing of the applied threshold. These are saved from each signal then checked for coincidence. If all signals from each pixel are above threshold, then the earliest time is stored. If one signal is not above threshold the entire run is discarded. The earliest signal from each end of the scintillator is then saved and used for statistics. The simulation will then reset for another event. The results of all the events are tallied and used to determine the timing resolution of the experiment. figure 3.3 is an example of the final output for the simulation. In addition to the final data being saved in '.root' files, a log of the simulation is stored in '.txt' for future analysis or debugging.

4. Data Acquisition

A large portion of the semester was spent on data acquisition and analysis. Starting with and building a new apparatus required setting up an acquisition system. Since time and resources were short, a simple oscilloscope with acquisition scripts was the appropriate choice. Basic communication scripts for the tds7104 were provided by. The scripts were modified and tailored to fit the needs of the project. This section describes how to use the new scripts for various acquisitions. These scripts have been successfully tested in Scientific Linux and Ubuntu 14.04LTS and should work in other versions.

4.1. Basic Operation

To run the new scripts simply download and compile the file provided (tekvisascript.zip). Navigate to the directory containing the zip file and unload in any directory one desires. Then use the terminal commands provided:

```
$ unzip tekvisascript.zip
$ cd tek_1.04/
```

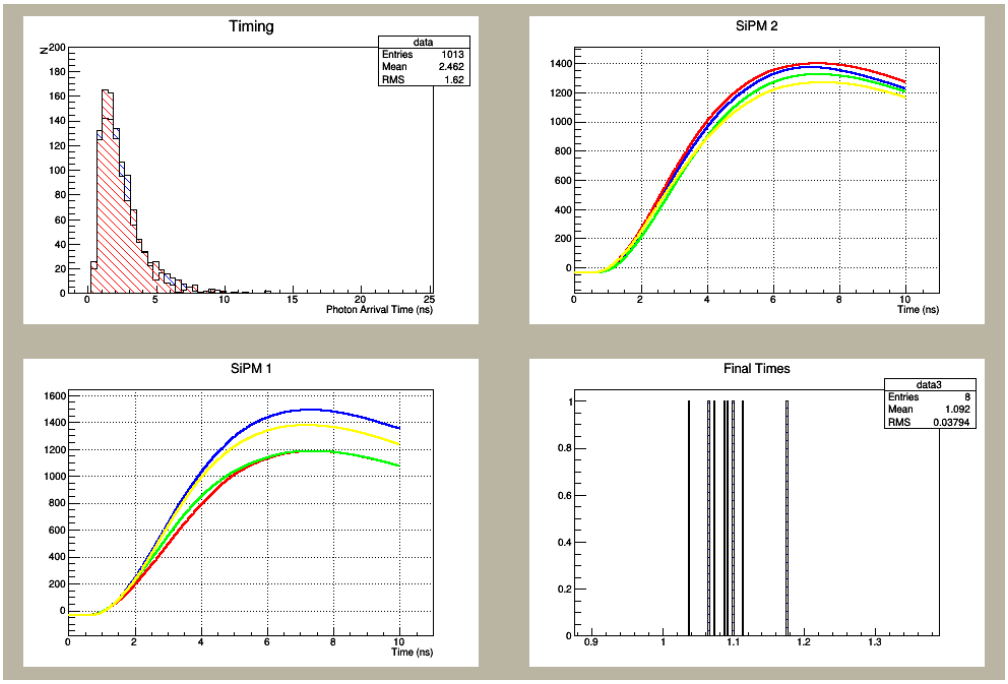


Figure 3.3: Displays a sample output of the simulator. The first graph shows all timings, the 2nd and 3rd show the final signals from each array, while the last shows the results of threshold timing.

```
$ make
```

If this compiles correctly and without any warning one can begin taking traces. To take data navigate to the /*tgetwf* directory and run the following commands

```
$ cd utils/tgetwf/
```

```
$ ./tgetwf -ip (ip adress of scope) -f (filename) -c (channel) -r (number of traces you wa
```

heres an example for clarity:

```
$ ./tgetwf -ip 142.90.104.79 -f Laz_120PE_26.5V -c 2 -r 1000
```

This should generates two files, .wf and .wfi which will be located in the /*tgetwf* directory by default. The '.wf' is the data binary file and '.wfi' is the information file which contains scaling and offset values needed to interpret the binary data correctly. Here are some additional notes to consider when taking data which these scripts: If you clip the waveform (vertical) in the trace NO data will be recorded/sent for that part, this will be important to consider if one wishes to preserve linear timing data. Included also is the (vxi11/) you can use this for sending individual commands to scope ie taking/sending measurements adjusting scope acquisition parameters etc...Instructions on how to set this up are in the scripts.

4.2. BINARY CONVERSION

First run trace graber(./tgetwf -ip your.ip.adress -f filename -c channel -r numruns), provided and as described above. Next convert binary file (.wf) to hex using unix/linux commmand:

```
$ hexdump -x < binaryfile.wf > hextextfile.txt
```

This command makes a file full of strings which can be sorted and converted into a float value. Each line should follow this format below, if it doesn't this string sorter won't work and need to make appropriate adjustments (DPO 7000 series was different)

```
0000000    4f00    5000    5100    5100    5300    5400    5100    5000
0000010    5100    5200    5300    5100    5000    5200    5000    5100
0000020    4f00    4f00    5000    4d00    5000    4e00    4f00    4d00
0000030    4f00    4f00    4d00    4d00    4f00    4d00    4c00    4e00
```

Each line should be 71 characters long, if not it will be dismissed, sometimes a line is emty or filled with garbage. The vxi-11/tekVISA binary transfer protocol is 2 byte 8 bit signed. These are formatted into a vector like this (first line from above):

```
 4f    50    51    51    53    54    51    50
```

Each value in the string is then converted to 8 bit signed integers, then converted to floats and adjusted based on parameters from the .wfi file provided by trace grabber like this. There are simpler methods that can be explored that would hopefully avoid using the hexdump command and import directly from the binary file.

```
for(int i = 0; i < hex.size(); ++i)
                {temp = hex.at(i);
                unsigned long int x;
                 std::stringstream ss;
                        ss << std::hex << temp;
                        ss >> x;
                int y = (int8_t) x;
                float f = ((256*y*vertgain)-vertoff)*1000;//into mV
                // output it as a signed type
                        fin.push_back(f);
                //cout << "\n" << f;//for debugging
                }
```

5. DECTECTOR TESTS

The electronic detector group was asked to design the front end electronics for the spectrometer, which required characterization of the devices of choice. One SiPM from sensl was selected

as the most viable option. Figure **??** shows the devices used and tested. At the time the low noise C-series 6mm device hadn't been released. In order to advance electronics design earlier models were characterized. Both the 3mm B-series and 6mm C series (non-lownoise) were characterized.

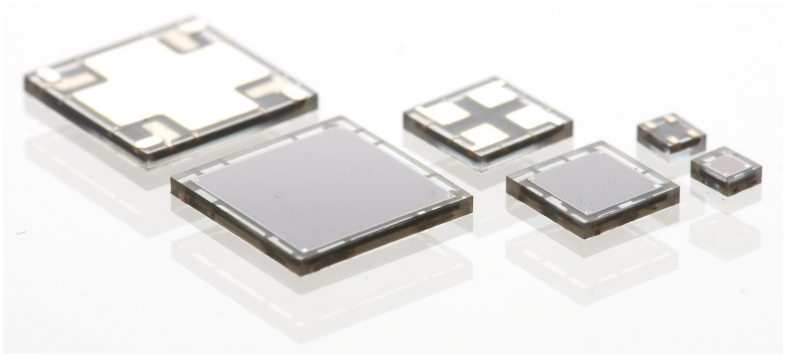


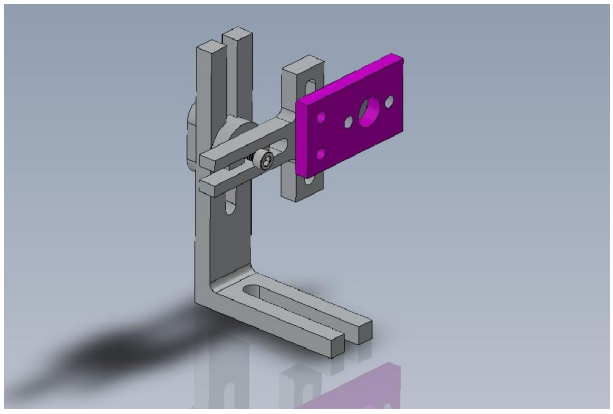
Figure 5.1: A picture of the devices used for detector tests; 6mm c-series and 3mm b-series.

5.1. Apparatus Design

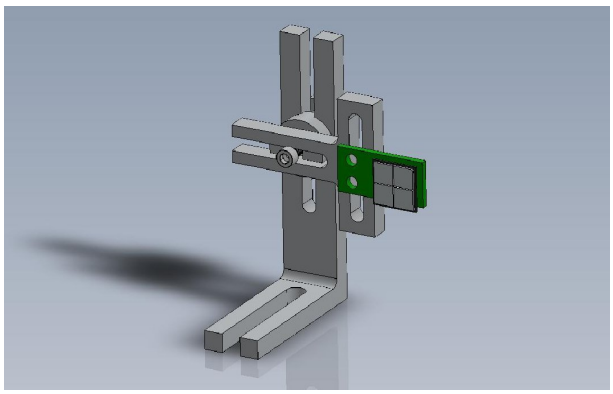
In order to perform detector tests a custom apparatus needed to be designed and fabricated. A slotted bracket and breadboard style design was selected for its adjustability and adaptability for different tests. Each piece was designed and fabricated at TRIUMF, the drawings are available in the appendix, or through TRIUMFs cad library (drawing numbers located in schematics). A slotted washer ensures level and straight adjustability which is shown in Figure 5.2. Full configurations for laser, source and device mounting is displayed in Figures 5.2 and 5.3. The final fabricated apparatus in operation is shown in figure 5.4.

5.2. Electronics

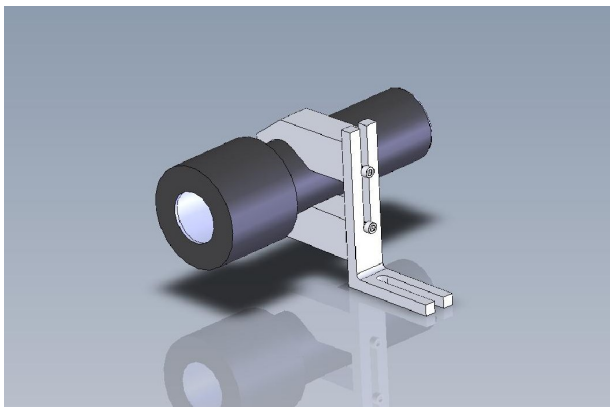
There are a number of ways to read out the pixels from each array. These can be summed into two categories, Independent and multiplexing. The independent method involves reading out each pixel into its own channel. This makes for excellent resolution however, adding channel increases costs, design complexity and data output. Additionally large circuitry is required to handle even simple arrays. Multiplexing is a method used to reduce complexity and cost however can reduce timing due to the increase in capacitance that comes with typical summing configurations. The goal is to find a healthy balance of channels, ie selecting the most one can attain given costs, timing and design constraints while minimizing the amount of multiplexing. This further shows the importance of testing the effects of multiple pixels summed together. In addition to simple summing configurations and basic biasing circuit was used for testing which originates from sensl recommended operating procedures which is displayed in figure 5.5.



(a)



(b)



(c)

Figure 5.2: Displays the many configurations the support pieces can take on the accommodate the parameters of detector tests

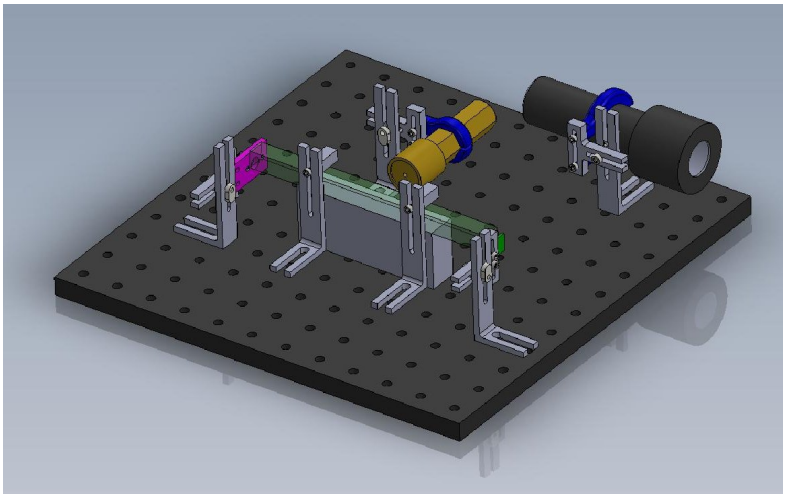


Figure 5.3: Displays a possible configuration for laser and source tests with base

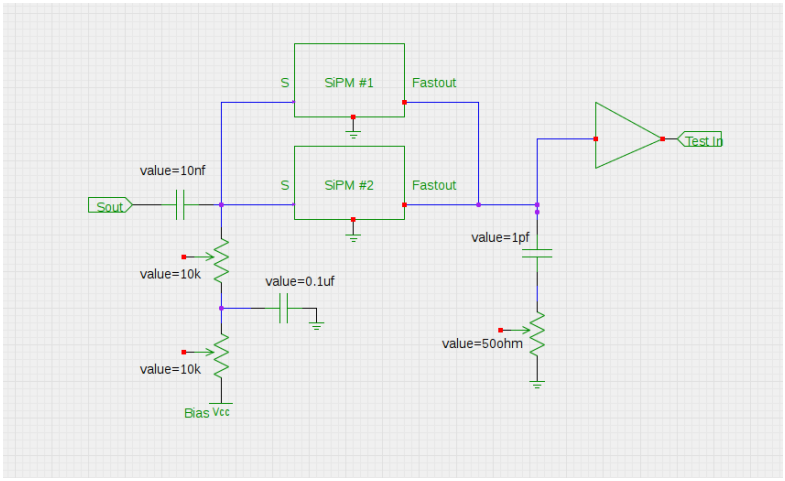


Figure 5.5: Displays the biasing configuration used for C and B series devices

5.3. Analysis

Data collected during testing was analyzed using ROOT which a standard analysis program common in particle physics. Below is an outline of the tests performed for N number of pixels and V_{ob} threshold levels:

1. SiPM/amplifier responce function
2. Dark counts vs. threshold
3. Time resolution with laser

The response function of the amplified was determined by injecting a short square pulse into the 'test' input. The step response of a dynamical system gives information on the stability of such a system, and on its ability to reach one stationary state when starting from another To

inject charge impulse you apply a voltage step from rectangular pulse from a pulse or function generator was a resolution of at least 80 ps. A 1 mV rectangle (step) will inject $1mV * 1pF = 1fC$. An SiPM pulse for the b-series is 2.6 fC per $V_{ob} = 1V$. The amplifier response is about 0.5mV/fC.

$$V(t) = V_{max}(exp(\frac{-(t-t_o)}{\tau_{Rise}}) - exp(\frac{-(t-t_o)}{\tau_{Fall}})) \tag{5.1}$$

Therefore a response of 10 fC was measured which gives approximately a 5mV peak, large enough to be measured with an oscilloscope. Its important to note the bias doesn't denote the impedance of the SiPM, and the step response is only dependent on the amplitude of the step and the network time constant, which is independent of V_{ob}, therefore the response should also be independent of V_{ob}. An example of response function of the amplified taken while connected to a B-series pixel is displayed in figure 5.6. The average response was fitted using root which enabled to calculation of the rise and fall times. The equation used to fit the response is displayed as equation 5.1.

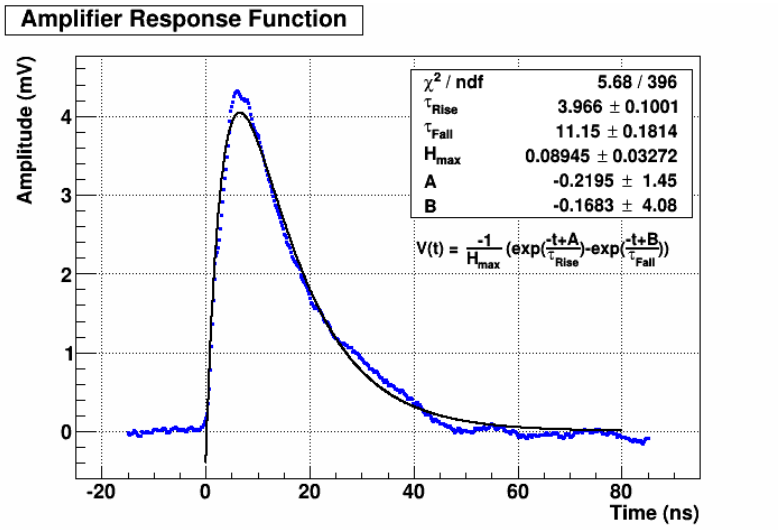
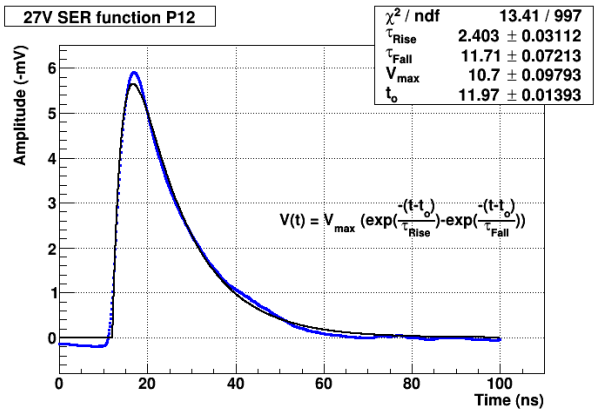
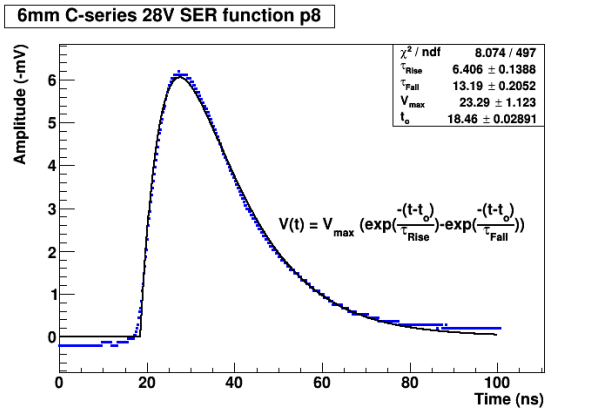


Figure 5.6: Displays the biasing configuration used for C and B series devices

A single photo-electric pulse was measured and fitted. This was done to characterize the rise and fall time of the SiPM which would give insight into the possible timing resolution of the spectrometer. 1000 signals were averaged and fitted which is displayed in 5.7



(a)

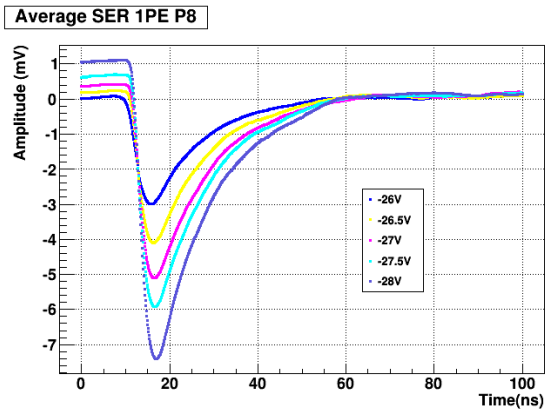


(b)

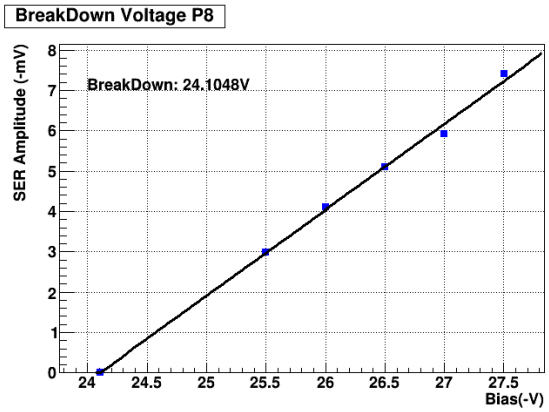
Figure 5.7: Displays the single photo-electric response of the 3mm B-series (a) and the 6mm C-series (b).

Additionally different number of pixels were summed together to gauge the affect of multiplexing. This is demonstrated in figure 5.8 which shows the single photo-electric response for 1, 2, 3, and 4 pixels. Its interesting to note the increase in rise and fall times which can be crucial in the timing performance of the spectrometer. One should notice the artifact before the pulse becomes more significant for additional pixels and ultimately discredits the fit. The fit was only found to be valid for single pixels.

The breakdown point of the pixel is also required to figure out the correct way to bias the detector. Breakdown was calculated by finding the relation between amplitude and V_{ob} and then solving for when amplitude is zero. This is shown in figure 5.9.



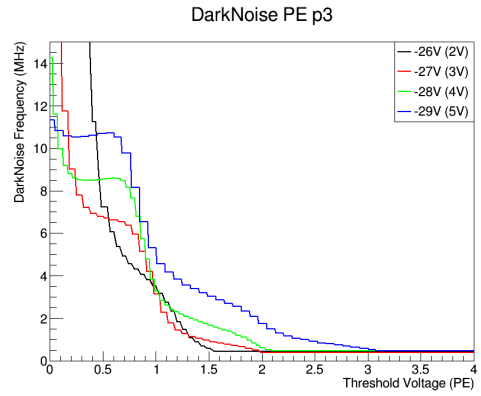
(a)



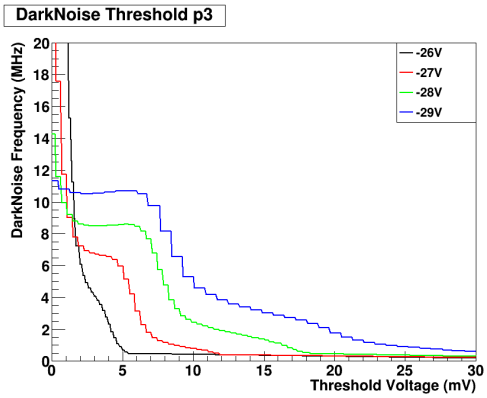
(b)

Figure 5.9: Shows the signals collected a) and used to calculate the breakdown voltage b) of the device

When detecting any signal noise to signal level is a crucial value to know. For SiPM's the noise can vary with bias and summing configuration. Using the tekvisa trace grabber, A 1000, 400ns traces were captured and analyzed for pulse frequency for 1,2,3 and 4 PE thresholds as well as for pixel multiplexing. Two different pulse detection methods were used; The first based solely on threshold, and the other based on pulse amplitude. Shown in figure 5.10 and figure 5.11 are some noise data renderings.



(a)



(b)

Figure 5.10: Displays an example of the darknoise data collected from a B-series 3mm pixel. part a) displays part b) normalized in photo-electrons.

The "trigger" charts show all signals with a specific amplitude at a particular threshold which exclude hits above an "upper limit", while the "threshold" will show all hits that occur at and above a specific threshold. This data can not only be used for determination of noise frequencies at different thresholds it can be used to help determine the range of particular amplitudes for different photo-electron levels.

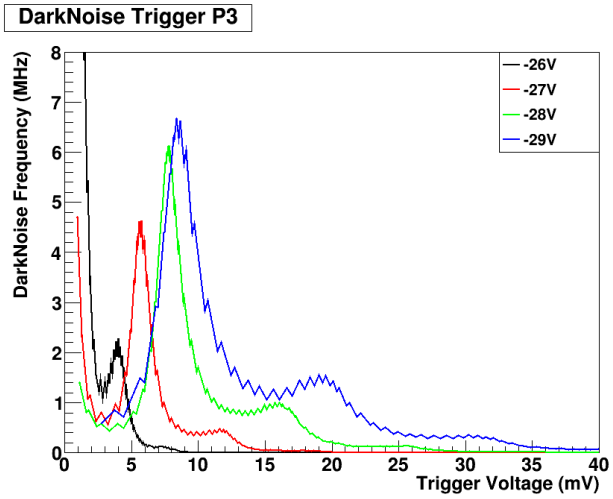


Figure 5.11: Shows all signals with a specific amplitude at a particular threshold which exclude hits above an "upper limit".

The conclusions of the noise tests on the final devices will help determine the ideal biasing levels and amplification. Additional tests included investigation of the intrinsic resolution of the SiPM's. The can be found by firing a short laser pulse 80ps into the device and looking at the consistency of the timing from the output signal. The spread of the timings ultimately determine the intrinsic timing of SiPM and amplifier which will exclude timing affects due to geometry and positron energy.

The conclusions reached signify that a 50ps resolution can be attained for the B-series given the energy of the positron is high enough (>100pe) to provide a high intensity and given an optimal bias threshold (27V) is used.

6. Future Work

The new low-noise C-series SiPM's planned for installment on the new M9 prototype muon spectrometer need to be fully characterized as was done with the 3mm B-series and 6mm C-series. Below lists the remaining tasks of the detector tests.

Prototype Goals:

- realistic scintillation transient to be used in simulations - time resolution with/without fluctuations of photon propagation

1a. "Small" scintillator - SPM type C 3x3 mm - SPM type C 6x6 mm - 1, 2 pixels per preamp - Sr-90 source and laser into opposite ends at verious angles? - recording waveforms with oscilloscope - calculate timing offline vs threshold and Vob Outcome: scintillation transient. Its effect on timing

1b. "Big" scintillator. - same tests as B1. Might need more pixels to check 3 and 4 pixels per preamp - compare 6x6 with four 3x3 per preamp Outcome: contribution of photons collection

2. Final design Goals: - characterization/commissioning of M9 counters - optimization of thresholds and bias

2a. Muon counter - electrical tests - laser tests - Alpha source (St-90 standard source not feasible for thin muon counter)

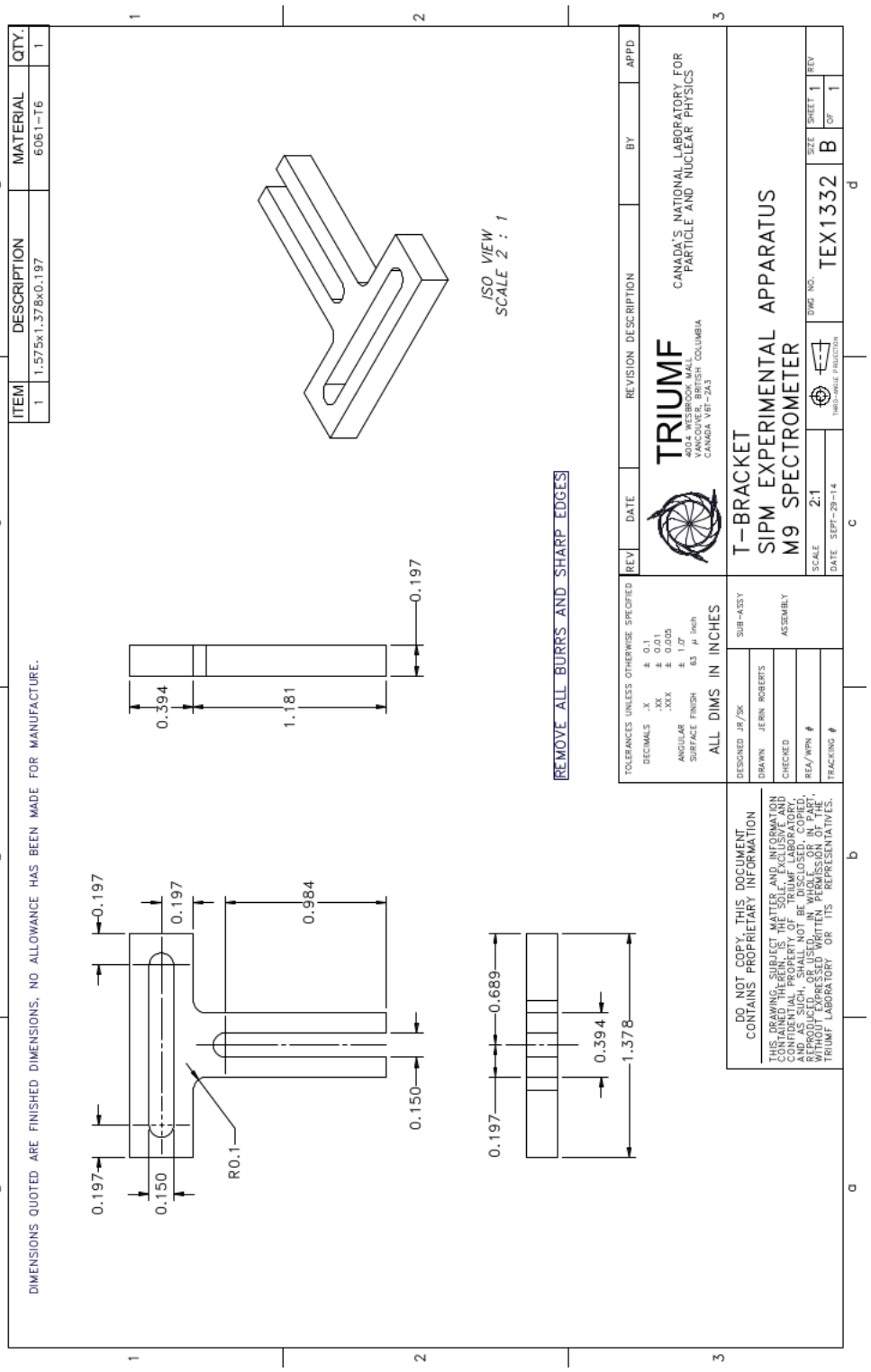
2b. Active collimator - Same standard tests

2c. Back positron counter - Same standard tests

2d. Transverse positron counter - Same standard tests

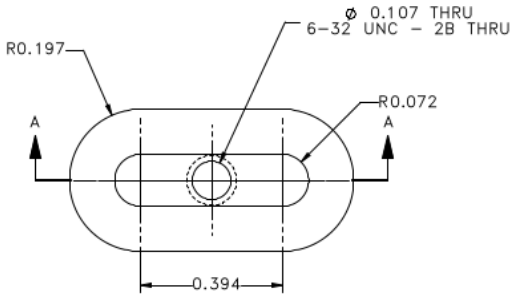
A. Appendix

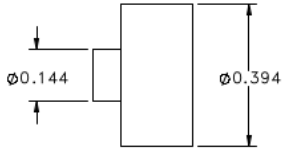
A.1. Drawings

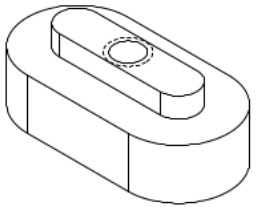


DIMENSIONS QUOTED ARE FINISHED DIMENSIONS, NO ALLOWANCE HAS BEEN MADE FOR MANUFACTURE.

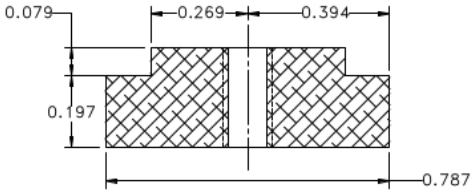
ITEM	DESCRIPTION	MATERIAL	QTY.
1	0.787x0.394x0.276LG.	6061-T6	1







ISO VEIW
SCALE 4 : 1



SECTION A-A

REMOVE ALL BURRS AND SHARP EDGES

TOLERANCES UNLESS OTHERWISE SPECIFIED
DECIMALS .X ± 0.1
.XX ± 0.01
.XXX ± 0.005
ANGULAR ± 1.0°
SURFACE FINISH 63 µ inch
ALL DIMS IN INCHES

REV	DATE	REVISION DESCRIPTION	BY	APPD

TRIUMF
4004 WESBROOK MALL
VANCOUVER, BRITISH COLUMBIA
CANADA V6T-2A3

CANADA'S NATIONAL LABORATORY FOR PARTICLE AND NUCLEAR PHYSICS

DESIGNED JR/SK
DRAWN JERIN ROBERTS
CHECKED
REA/WPN #
TRACKING #

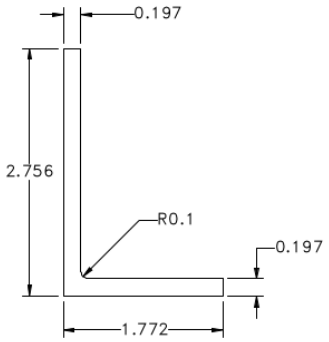
SUB-ASSY
ASSEMBLY

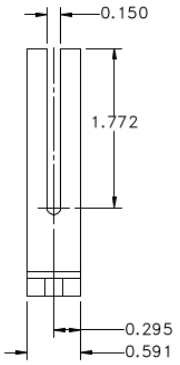
L-BRACKET NUT
SIPM EXPERIMENTAL APPARATUS
M9 SPECTROMETER

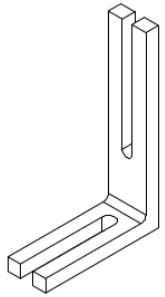
SCALE 4:1
DATE SEPT-29-14
THIRD-ANGLE PROJECTION
DWG NO. TEX1334
SIZE B
SHEET 1 OF 1
REV

DIMENSIONS QUOTED ARE FINISHED DIMENSIONS, NO ALLOWANCE HAS BEEN MADE FOR MANUFACTURE.

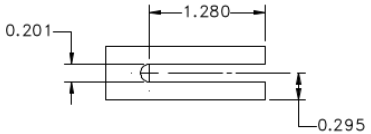
ITEM	DESCRIPTION	MATERIAL	QTY.
1	2.756x1.772x0.591LG.	6061–T6	1







ISO VIEW
SCALE 1 : 1



REMOVE ALL BURRS AND SHARP EDGES

TOLERANCES UNLESS OTHERWISE SPECIFIED
DECIMALS .X ± 0.1
.XX ± 0.01
.XXX ± 0.005
ANGULAR ± 1.0°
SURFACE FINISH 63 μ inch
ALL DIMS IN INCHES

REV	DATE	REVISION DESCRIPTION	BY	APPD

TRIUMF
4004 WESBROOK MALL
VANCOUVER, BRITISH COLUMBIA
CANADA V6T–2A3

CANADA'S NATIONAL LABORATORY FOR PARTICLE AND NUCLEAR PHYSICS

DESIGNED JR/SK
DRAWN JERIN ROBERTS
CHECKED
REA/WPN #
TRACKING #

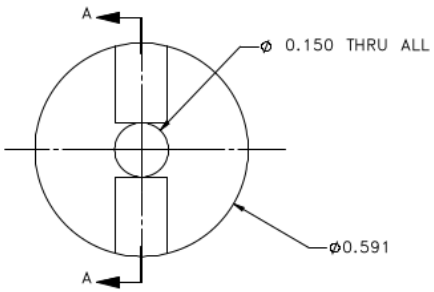
SUB–ASSY
ASSEMBLY

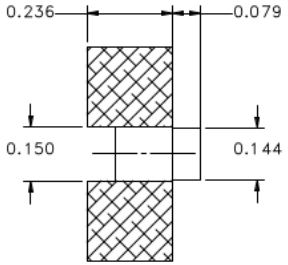
L–BRACKET
SIPM EXPERIMENTAL APPARATUS
M9 SPECTROMETER

SCALE 1:1
DATE SEPT–29–14
THIRD–ANGLE PROJECTION
DWG NO. TEX1331
SIZE B
SHEET 1 OF 1
REV

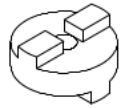
DIMENSIONS QUOTED ARE FINISHED DIMENSIONS, NO ALLOWANCE HAS BEEN MADE FOR MANUFACTURE.

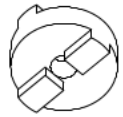
ITEM	DESCRIPTION	MATERIAL	QTY.
1	DIA. 0.295 X 0.315 LG.	6061-T6	1



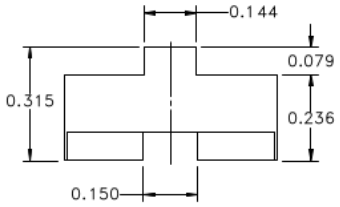


SECTION A-A





ISO VIEW
SCALE 2:1



REMOVE ALL BURRS AND SHARP EDGES

TOLERANCES UNLESS OTHERWISE SPECIFIED
DECIMALS .X ± 0.1
.XX ± 0.01
.XXX ± 0.005
ANGULAR ± 1.0°
SURFACE FINISH 63 μ inch
ALL DIMS IN INCHES

REV	DATE	REVISION DESCRIPTION	BY	APPD

TRIUMF
4004 WESBROOK MALL
VANCOUVER, BRITISH COLUMBIA
CANADA V6T-2A3

CANADA'S NATIONAL LABORATORY FOR PARTICLE AND NUCLEAR PHYSICS

DESIGNED JR/SK
DRAWN JERIN ROBERTS
CHECKED
REA/WPN #
TRACKING #
SUB-ASSY
ASSEMBLY

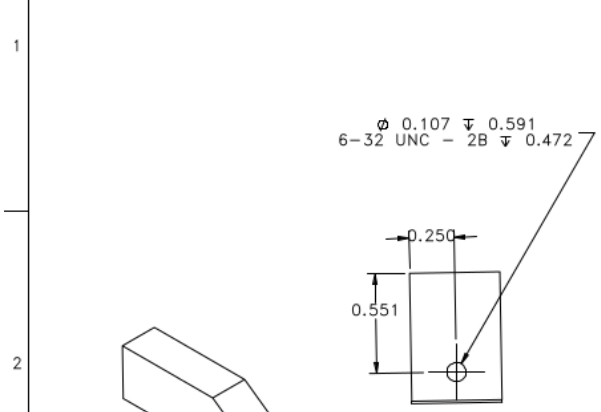
L-BRACKET MOUNT GUIDE
SIPM EXPERIMENTAL APPARATUS
M9 SPECTROMETER

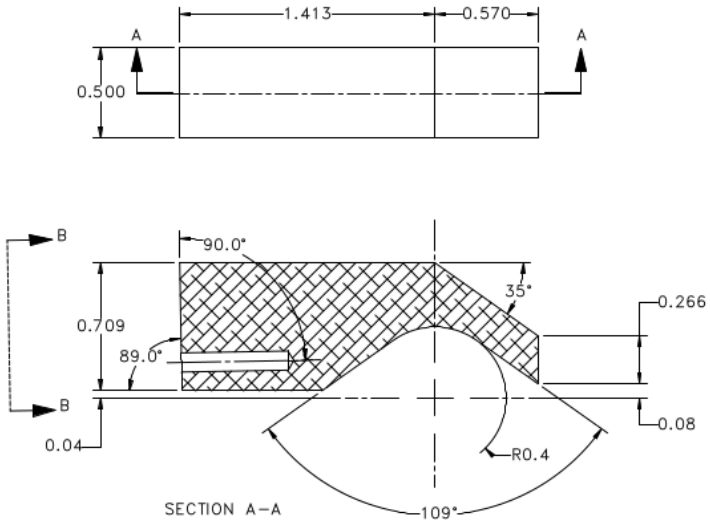
SCALE 4:1
DATE SEPT-29-14
THIRD-ANGLE PROJECTION
DWG NO. TEX1333
SIZE B
SHEET 1 OF 1
REV

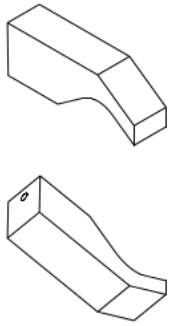
DIMENSIONS QUOTED ARE FINISHED DIMENSIONS, NO ALLOWANCE HAS BEEN MADE FOR MANUFACTURE.

REMOVE ALL BURRS AND SHARP EDGES

ITEM	DESCRIPTION	MATERIAL	QTY.
1	0.709x0.591x1.983LG	6061-T6	4







ISO VIEW
SCALE 1 : 1

TOLERANCES UNLESS OTHERWISE SPECIFIED

DECIMALS .X ± 0.1
.XX ± 0.01
.XXX ± 0.005
ANGULAR ± 1.0°
SURFACE FINISH 63 µ inch

ALL DIMS IN INCHES

REV	DATE	REVISION DESCRIPTION	BY	APPD

TRIUMF
4004 WESBROOK MALL
VANCOUVER, BRITISH COLUMBIA
CANADA V6T-2A3

CANADA'S NATIONAL LABORATORY FOR PARTICLE AND NUCLEAR PHYSICS

DESIGNED JR/SK
DRAWN JERIN ROBERTS
CHECKED
REA/WPN #
TRACKING #

SUB-ASSY
ASSEMBLY

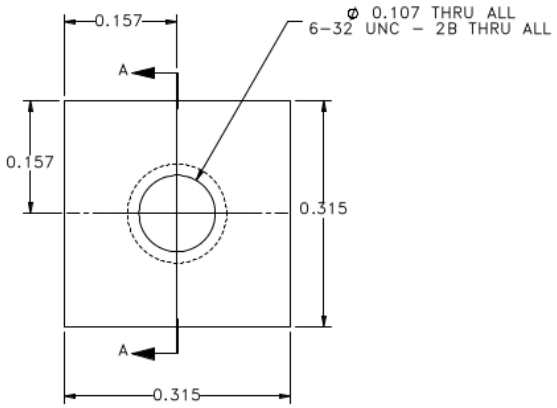
MISC. CLAMP
EXPERIMENTAL APPARATUS
M9 SPECTROMETER

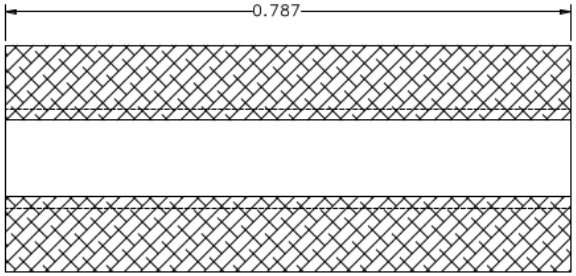
SCALE 2:1
DATE OCTOBER 15, 2014
THIRD-ANGLE PROJECTION

DWG NO. TEX1337
SIZE B
SHEET 1 OF 1
REV

DIMENSIONS QUOTED ARE FINISHED DIMENSIONS, NO ALLOWANCE HAS BEEN MADE FOR MANUFACTURE.

ITEM	DESCRIPTION	MATERIAL	QTY.
1	0.315X0.315x0.787LG.	6061–T6	1





SECTION A–A

REMOVE ALL BURRS AND SHARP EDGES

TOLERANCES UNLESS OTHERWISE SPECIFIED
DECIMALS .X ± 0.1
.XX ± 0.01
.XXX ± 0.005
ANGULAR ± 1.0°
SURFACE FINISH 63 μ inch
ALL DIMS IN INCHES

REV	DATE	REVISION DESCRIPTION	BY	APPD

TRIUMF
4004 WESBROOK MALL
VANCOUVER, BRITISH COLUMBIA
CANADA V6T–2A3

CANADA'S NATIONAL LABORATORY FOR PARTICLE AND NUCLEAR PHYSICS

DESIGNED JR/SK
DRAWN JERIN ROBERTS
CHECKED
REA/WPN #
TRACKING #

SUB–ASSY
ASSEMBLY

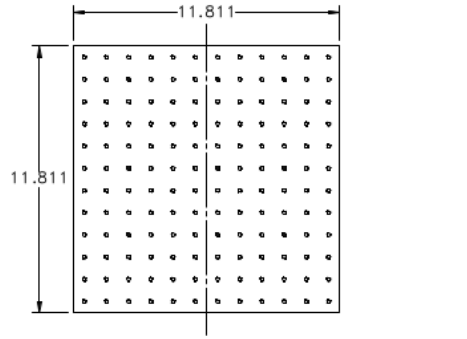
L–BRACKET BLOCK MOUNT
SIPM EXPERIMENTAL APPARATUS
M9 SPECTROMETER

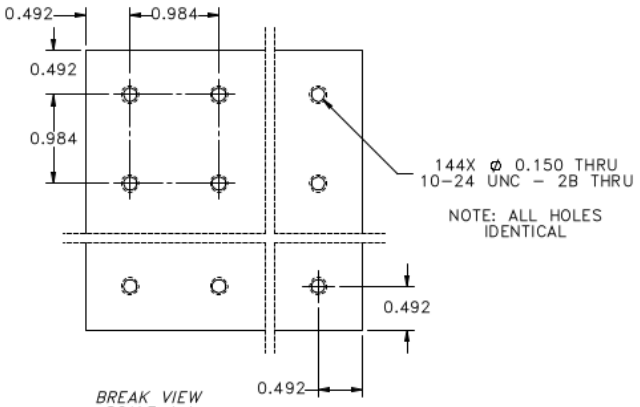
SCALE 8:1
DATE SEPT–29–14
THIRD–ANGLE PROJECTION
DWG NO. TEX1336
SIZE B
SHEET 1 OF 1
REV

DIMENSIONS QUOTED ARE FINISHED DIMENSIONS, NO ALLOWANCE HAS BEEN MADE FOR MANUFACTURE.

ITEM	DESCRIPTION	MATERIAL	QTY.
1	11.811x11.811x0.500	Al TOOLING PLATE	1

NOTE: AL TOOLING PLATE CAN BE SUPPLIED IF NECESSARY





BREAK VIEW
SCALE 1:1

REMOVE ALL BURRS AND SHARP EDGES

TOLERANCES UNLESS OTHERWISE SPECIFIED
DECIMALS .X ± 0.1
.XX ± 0.01
.XXX ± 0.005
ANGULAR ± 1.0°
SURFACE FINISH 63 μ inch
ALL DIMS IN INCHES

REV	DATE	REVISION DESCRIPTION	BY	APPD

TRIUMF
4004 WESBROOK MALL
VANCOUVER, BRITISH COLUMBIA
CANADA V6T-2A3

CANADA'S NATIONAL LABORATORY FOR PARTICLE AND NUCLEAR PHYSICS

DESIGNED JR/SK
DRAWN JERIN ROBERTS
CHECKED
REA/WPN #
TRACKING #
SUB-ASSY
ASSEMBLY

BASE PLATE
SIPM EXPERIMENTAL APPARATUS
M9 SPECTROMETER

SCALE 1:4
DATE SEPT-29-14
THIRD-ANGLE PROJECTION
DWG NO. TEX1335
SIZE B
SHEET 1 OF 1
REV

A.2. Data Visualizer Code

```
//Data Visualizer using ROOT and Tekvisa trace grabber
simplified with no peak detection

#include <fstream>
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <TLatex.h>
#include <TImage.h>
#include "TF1.h"
#include "TH2F.h"
#include <TRandom3.h>
#include <TAxis.h>
#include <TH1F.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <cstdlib>
#include <TPad.h>
#include <TFrame.h>
#include <cmath>
#include <TGraph2D.h>
#include <TGraph.h>
#include "TROOT.h"
#include "TSystem.h"
#include "TMath.h"
#include "TFile.h"
#include <TMultiGraph.h>
#include <TLegend.h>
using namespace std;

template <typename T> string tostr(const T& t) {
   ostringstream os;
   os<<t;
   return os.str();
}

int main()
{

TCanvas *c1 = new TCanvas("c1","The FillRandom example",200,10,900,700);
const char * c;
float vertgain, vertoff, horstep, thres, uthres;

char* search = "% Vertical gain:";
char* search2 = "% Vertical offset:";
char* search3 = "% Horizontal interval:";
```

```
string titlefile2 = "Data Raw; Time (ns); Amplitude (mV)";



        //enter .txt and .wfi files to be used
string wfifile= "data/noisedata/p3_10000_dn26V.wfi";
string datafile= "data/noisedata/p3_10000_dn26V.txt";
c = wfifile.c_str();

ifstream file(c);
string str, temp, tomp;
string::size_type sz;
while (getline(file, str))
 { if (str.find(search, 0) != string::npos)
 {  getline(file, str);
  vertgain = ::atof(str.c_str());
cout << "\n" << "found verticle gain: " << vertgain;
     }

   if (str.find(search2, 0) != string::npos)
 {  getline(file, str);
  vertoff = ::atof(str.c_str());
cout << "\n" << "found verticle offset: " << vertoff;
     }

    if (str.find(search3, 0) != string::npos)
 {  getline(file, str);
  horstep = (::atof(str.c_str()))*1.0e09;
cout << "\n" << "found horizontal step: " << horstep;
     }
 }

file.close();




float lat = 6;
c = datafile.c_str();
//import and sort hex**********************
cout << "\nimporting strings\n";
vector<string> lines, hex;
vector<float> fin;
ifstream file2(c);
while (getline(file2, str))
{ if(str.length()==71)
{
```

```
str.erase (str.begin(), str.begin()+11);
for(int i = 1; i < 8; ++i)
{
str.erase (str.begin()+(i*2), str.begin()+(i*2+6));
}
str.erase (str.end()-2, str.end());
lines.push_back(str);
}
}
cout << "done\n";
cout << "dividing lines\n";
for(int i = 0; i < lines.size(); ++i)
{temp = lines.at(i);
for(int i = 1; i < 9; ++i)
{tomp = temp;
tomp.erase (tomp.begin()+(i*2), tomp.end());
if(i!=1)
tomp.erase (tomp.begin(), tomp.begin()+(i*2-2));
hex.push_back(tomp);
}
}
cout << "done\n";
cout << "converting hex\n";
 file2.close();


/*for(int i = 0; i < lines.size(); ++i)
{temp = lines.at(i);

cout << temp << "\n";
}*/

for(int i = 0; i < hex.size(); ++i)
{temp = hex.at(i);
unsigned long int x;
  std::stringstream ss;
ss << std::hex << temp;
ss >> x;
int y = (int8_t) x;
float f = ((256*y*vertgain)-vertoff)*1000;
// output it as a signed type
fin.push_back(f);
//cout << "\n" << f;
}
cout << "done\n";


c1->cd();
```

```
float x,y;
float tmp;
TGraph *gr = new TGraph();
for (float i = 0; i < fin.size(); ++i)
{
x = i*horstep;
tmp = fin.at(i);
y = tmp;
gr->SetPoint(i,x,y);
}

c = titlefile2.c_str();
gr->SetTitle(c);
        gr->Draw();


}
```

A.3. SIMULATION CODE

```
#include "TF1.h"
#include "TH2F.h"
#include <TRandom3.h>
#include <TAxis.h>
#include <TH1F.h>
#include <vector>
#include <TStyle.h>
#include <TCanvas.h>
#include <iostream>
#include <cstdlib>
#include <TPad.h>
#include <TFrame.h>
#include <cmath>
#include <TGraph2D.h>
#include <TGraph.h>
#include "TROOT.h"
#include "TSystem.h"
#include "TMath.h"


using namespace std;




int main() {
```

```
cout << "starting...\n";
TRandom *gRandom = new TRandom3();
TFormula *form1 = new TFormula("form1","-(exp(-x/[0])-exp(-x/[1]))");

form1->SetParameter(0, .5);
form1->SetParameter(1, 1.5);
TF1 *sqroot = new TF1("sqroot","form1",0,15);
sqroot->SetNpx(500);
int np = 50000;//number of photons generated
//SiPM parameters
float pixsize = 0.000035;// in meters
float deadtime = 30;//in nano seconds
//variables for geometry (sqaure scintillator piece)
        float length = .15;
float stlength = 0.0;
float rad1 = .035;
float rad2 = 0.050;
        float tmp = 0;
        float stepsize = 0.00001;
float PEthres = 3;
float cang = 0.30, cang2 = 0.78;
if (stlength >= (length/2)){
cout <<"\nstart position error";
}
//vectors
//..............with geometry
vector<float> rand (np*3);//......random directions
vector<float> scintp (np);
vector<float> lpath, lpath2;

vector< vector<float> > finalpos, finalpos2, nan, startpos;


//Generate Photon Paths, return times
cout << "generating random photons...\n";
float tomp;//temporary
int nans =0, xtalk=0, xtalk2=0, p = 0;

//this generates random components for direction vecto


            for (float i = 0; i < rand.size(); ++i)
             {tomp = gRandom->Uniform(-10,10);
                rand.at(i)=tomp;
             }

std::vector<float> scintx (np), scinty (np), scintz (np);
float zomp, jomp, xs, ys, radt, radts;
//tomp = tan(gRandom->Uniform(0.698,0.8726));//slope ranges NO NEGS
```

```
 tomp = tan(0.78);
// zomp = gRandom->Uniform(-0.087,0.087);//angle range (rads)
 zomp = 0.0;

cout << "generating random start locations...\n";
for (float w = 0; w < np; ++w)
{

xs = gRandom->Uniform(0,10);
ys = tomp*xs;
radt = sqrt(pow(xs,2) + pow(ys,2));
radts=((rad1)+(radt*((rad2-rad1)/sqrt(200))));
scintz.at(w) = radts*tan(zomp) + stlength;
jomp = radt/radts;
scintx.at(w) = xs/jomp;
scinty.at(w) = ys/jomp;
bool out = (rad2 <= sqrt(pow(scintx.at(w),2) + pow(scinty.at(w),2)));
bool out2 = (rad1 >= sqrt(pow(scintx.at(w),2) + pow(scinty.at(w),2)));
if(scintz.at(w) >= (length/2))
{scintz.at(w) = 0;
cout << "\n ***ERROR : Solid angle beyond scintillator Length****";
}
if(out || out2)
--w;
}


//Path generator Loop
for(int n = 0; n < rand.size(); n = n+3)
{
vector<float> pos (3);//position vector (xyz)
vector<float> dir (3), stdir (3);//direction vector
vector<float> v1 (3);
    vector<float> v2 (3);
vector<float> normalv (3);

//variables
        float norml,fprime,magdir;
float pl = 0;
bool run = true;//true if photon hasn't reached end of scintillator

bool sensArea = false;
bool sensArea2 = false;//true if photon hits SiPM

//starting point of photon
pos.at(0) = scintx.at(p);//x
pos.at(1) = scinty.at(p);//y
pos.at(2) = scintz.at(p);//z
```

```
//random starting direction of photon
dir.at(0) = rand.at(n);
dir.at(1) = rand.at(n+1);
dir.at(2) = rand.at(n+2);
// to insure no infinite paths are generated
        if(dir.at(2) < 0.1 && dir.at(2) > -0.1 )
{
dir.at(2) = 0.1;
}

//now need to normalized direction vector

norml = sqrt((pow(dir.at(0),2)+pow(dir.at(1),2)
+pow(dir.at(2),2))/pow(stepsize,2));

for (float i = 0; i < dir.size(); ++i)
{
dir.at(i)= (dir.at(i)/norml);
}
startpos.push_back(pos);

//begin individual path generation
while(run)
{

//If encounters outer radial surface
 if (rad2 <= sqrt(pow(pos.at(1),2)+pow(pos.at(0),2)))
 {
if(rad2-pos.at(0) <=0)

{
dir.at(0) = -dir.at(0);

}
else
{
fprime = -(pos.at(0))/(sqrt(pow(sqrt(pow(pos.at(0),2)
+pow(pos.at(1),2)),2)-pow(pos.at(0),2)));

//put into vector:
v1.at(0) = 1;
v1.at(1) = fprime;
v1.at(2) = 0;
//second vector parallel to z axis
v2.at(0) = 0;
v2.at(1) = 0;
v2.at(2) = 1;

//cross product of v1 and v2 which gives normal vector of reflecting plane
```

```
normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
//check if crital angle is met
magdir = sqrt(pow(dir.at(0),2)
+pow(dir.at(1),2)+pow(dir.at(2),2));//magnitude of direction vector
//calc the magnitude of normal vector describing the reflecting plane
float magnormalv = sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2));
float modnormalv = pow(magnormalv,2);
//dot product between incident phonon direction vector and reflecting plane normal vector
float VdotN = dir.at(0)*normalv.at(0)
+dir.at(1)*normalv.at(1)+dir.at(2)*normalv.at(2);
//calc angle between inident photon and reflecting plane for criticle angle check.
float theta = acos(VdotN/(magdir*magnormalv));

if((theta <= cang) || (theta <= 0.78 && theta > 0.77))//critical angle check
{
run = false;
}
else
{

for (float i = 0; i < dir.size(); ++i)
{
 dir.at(i) = dir.at(i)-((2*normalv.at(i)*VdotN)/modnormalv);
}

norml = sqrt((pow(dir.at(0),2)/pow(stepsize,2))
+(pow(dir.at(1),2)/pow(stepsize,2))+(pow(dir.at(2),2)/pow(stepsize,2)));

for (float i = 0; i < dir.size(); ++i)
{
dir.at(i)= (dir.at(i)/norml);
}
}
}

}
//Same as above but for inner radial surface
if (rad1 >= sqrt(pow(pos.at(1),2)+pow(pos.at(0),2)))
 {

if(pos.at(0)-rad1 <=0 && pos.at(1) <=0)
{
dir.at(0) = -dir.at(0);
}
else
{
```

```
fprime = -(pos.at(0))/(sqrt(pow(sqrt(pow(pos.at(0),2)
+pow(pos.at(1),2)),2)-pow(pos.at(0),2)));
v1.at(0) = 1;
v1.at(1) = fprime;
v1.at(2) = 0;

v2.at(0) = 0;
v2.at(1) = 0;
v2.at(2) = -1;//the only thing thats different:



normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
//check if crital angle is met
magdir = sqrt(pow(dir.at(0),2)+pow(dir.at(1),2)+pow(dir.at(2),2));
float magnormalv = sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2));
float modnormalv = pow(sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2)),2);
float VdotN = dir.at(0)*normalv.at(0)
+dir.at(1)*normalv.at(1)+dir.at(2)*normalv.at(2);
float theta = acos(VdotN/(magdir*magnormalv));

if((theta <= cang) || (theta <= 0.78 && theta > 0.77))//critical angle check
{
run = false;
}
else
{

for (float i = 0; i < dir.size(); ++i)
{
 dir.at(i) = dir.at(i)-((2*normalv.at(i)*VdotN)/modnormalv);
}

norml = sqrt((pow(dir.at(0),2)/pow(stepsize,2))
+(pow(dir.at(1),2)/pow(stepsize,2))
+(pow(dir.at(2),2)/pow(stepsize,2)));

for (float i = 0; i < dir.size(); ++i)
{
dir.at(i)= (dir.at(i)/norml);
}
}
}
}
//for contact of verticle edge planes
```

```
if ( pos.at(1) >= (2*pos.at(0)))
 {//45 geometry

fprime = 2;
v1.at(0) = 1;
v1.at(1) = fprime;
v1.at(2) = 0;

v2.at(0) = 0;
v2.at(1) = 0;
v2.at(2) = 1;


normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
//check if crital angle is met
magdir = sqrt(pow(dir.at(0),2)+pow(dir.at(1),2)+pow(dir.at(2),2));
float magnormalv = sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2));
float modnormalv = pow(sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2)),2);
float VdotN = dir.at(0)*normalv.at(0)
+dir.at(1)*normalv.at(1)+dir.at(2)*normalv.at(2);
float theta = acos(VdotN/(magdir*magnormalv));

if((theta <= cang) || (theta <= 0.78 && theta > 0.77))//critical angle check
{
run = false;
}
else
{

for (float i = 0; i < dir.size(); ++i)
{
 dir.at(i) = dir.at(i)-((2*normalv.at(i)*VdotN)/modnormalv);
}

norml = sqrt((pow(dir.at(0),2)/pow(stepsize,2))
+(pow(dir.at(1),2)/pow(stepsize,2))+(pow(dir.at(2),2)/pow(stepsize,2)));

for (float i = 0; i < dir.size(); ++i)
{
dir.at(i)= (dir.at(i)/norml);
}

}

}
```

```
if (pos.at(1) <= ((0.5)*pos.at(0)))
 {//45 geometry

fprime = 0.5;
v1.at(0) = 1;
v1.at(1) = fprime;
v1.at(2) = 0;

v2.at(0) = 0;
v2.at(1) = 0;
v2.at(2) = 1;


normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
//check if crital angle is met
magdir = sqrt(pow(dir.at(0),2)+pow(dir.at(1),2)+pow(dir.at(2),2));
float magnormalv = sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2));
float modnormalv = pow(sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2)),2);
float VdotN = dir.at(0)*normalv.at(0)
+dir.at(1)*normalv.at(1)+dir.at(2)*normalv.at(2);
float theta = acos(VdotN/(magdir*magnormalv));

if((theta <= cang) || (theta <= 0.78 && theta > 0.77))//critical angle check
{
run = false;
}
else
{

for (float i = 0; i < dir.size(); ++i)
{
 dir.at(i) = dir.at(i)-((2*normalv.at(i)*VdotN)/modnormalv);
}

norml = sqrt((pow(dir.at(0),2)/pow(stepsize,2))
+(pow(dir.at(1),2)/pow(stepsize,2))+(pow(dir.at(2),2)/pow(stepsize,2)));

for (float i = 0; i < dir.size(); ++i)
{
dir.at(i)= (dir.at(i)/norml);
}

}
```

```
}

//reflection off far end



//calc dist traveled for one step and add it to tally
                magdir = sqrt(pow(dir.at(0),2)+pow(dir.at(1),2)+pow(dir.at(2),2));
pl = pl+magdir;

//adding direction vector to position ie moving a step
        for (float i = 0; i < dir.size(); ++i)
{
pos.at(i)= (dir.at(i)+pos.at(i));
}
//check to see if at end of scintillator
bool end1, end2;
end1 = pos.at(2) >= (length/2);
end2 = pos.at(2) <= (-length/2);
if(end1)
{
run = false;
xs = gRandom->Uniform(0,100);


v1.at(0) = 0;
v1.at(1) = 1;
v1.at(2) = 0;

v2.at(0) = 1;
v2.at(1) = 0;
v2.at(2) = 0;


normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
//check if crital angle is met
magdir = sqrt(pow(dir.at(0),2)+pow(dir.at(1),2)+pow(dir.at(2),2));
float magnormalv = sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2));
float VdotN = dir.at(0)*normalv.at(0)
+dir.at(1)*normalv.at(1)+dir.at(2)*normalv.at(2);
float theta = acos(VdotN/(magdir*magnormalv));

if(theta <= cang2 && xs <= 25 )
{
if(pos.at(1) <= (pos.at(0) + 0.010) && pos.at(1) >= (pos.at(0) - 0.010))
```

```
sensArea = true;

}



}
if(end2)
{
run = false;
xs = gRandom->Uniform(0,100);

//check to see if hit SiPM(boundary defined by two lines and radial surfaces
v1.at(0) = 1;
v1.at(1) = 0;
v1.at(2) = 0;

v2.at(0) = 0;
v2.at(1) = 1;
v2.at(2) = 0;


normalv.at(0) = (v2.at(1)*v1.at(2))-(v2.at(2)*v1.at(1));
normalv.at(1) = (v2.at(2)*v1.at(0))-(v2.at(0)*v1.at(2));
normalv.at(2) = (v2.at(0)*v1.at(1))-(v2.at(1)*v1.at(0));
//check if crital angle is met
magdir = sqrt(pow(dir.at(0),2)+pow(dir.at(1),2)+pow(dir.at(2),2));
float magnormalv = sqrt(pow(normalv.at(0),2)
+pow(normalv.at(1),2)+pow(normalv.at(2),2));
float VdotN = dir.at(0)*normalv.at(0)
+dir.at(1)*normalv.at(1)+dir.at(2)*normalv.at(2);
float theta = acos(VdotN/(magdir*magnormalv));

if(theta <= cang2 && xs <= 25 )
{
if(pos.at(1) <= (pos.at(0) + 0.010) && pos.at(1) >= (pos.at(0) - 0.010))
sensArea2 = true;
}



}

}//end of while


if(pl != pl)//prevent nans, only get about 3 in a million
{
rand.at(n) = rand.at(n) +0.01;
```

```
rand.at(n+1) = rand.at(n+1) +0.01;
rand.at(n+2) = rand.at(n+2) +0.01;
n = n-3;
++nans;
--p;
}
else
{
if(sensArea || sensArea2)
{
//log pathlength(time) and final position
if(sensArea){
lpath.push_back (pl);
finalpos.push_back(pos);
}
if(sensArea2){
lpath2.push_back (pl);
finalpos2.push_back(pos);
}
/*Cross talk generator: each neighbor excitation can lead to many others defined by
 prob of avalanche exciting neighbor
setup so a crosstalk event can also excite its neighbor with same prob ie a chain
can occur
(double hits on same pix taken care of later)*/
float fomp = gRandom->Uniform(0,100);//random num
while(fomp < 20)//defines prob of exciting one neighbor
{
//generates random direction to increment (one pixel size) outside original pixel
float x = gRandom->Uniform(-1,1);
float y = gRandom->Uniform(-1,1);
//normalize for pixel size
float A = sqrt(pow(x/(pixsize),2)+pow(y/(pixsize),2));
x = x/A;
y = y/A;
//apply change to last final position
pos.at(0) = pos.at(0) + x;
pos.at(1) = pos.at(1) + y;
//check to make sure still on SiPM
bool ta, tb, tc, td;
ta = (pos.at(1) <= (pos.at(0) + 0.010));
tb = (pos.at(1) >= (pos.at(0) - 0.010));
tc = (rad2 >= sqrt(pow(pos.at(1),2)+pow(pos.at(0),2)));
td = (rad1 <= sqrt(pow(pos.at(1),2)+pow(pos.at(0),2)));
if( ta && tb && tc && td)
{
//if true records same pathlength(time) and new final position
if(sensArea){
lpath.push_back (pl);
finalpos.push_back(pos);
```

```
++xtalk;
}
if(sensArea2){
lpath2.push_back (pl);
finalpos2.push_back(pos);
++xtalk2;
}

fomp = gRandom->Uniform(0,100);
}
else//stop new excitations if photon leaves SiPM
{fomp = 100;}
   }

dir.clear();
pos.clear();
v1.clear();
v2.clear();
}

}//end of else

++p;
}//end of for

cout << "\n Nans Generated: " << nans;

int ls = lpath.size();
int ls2 = lpath2.size();

//random path lengths
vector<float> pathtime (ls), pathtime2 (ls2);//random path times
vector<float> scint (ls), scint2 (ls2);//timing due to scintillation
vector<float> NGtime (ls), ftime (ls), ftime2 (ls2);//final timings

cout << "\ndone!\n";
//

for (float i = 0; i < ls; ++i)
{
pathtime.at(i) = (lpath.at(i)/0.2);
}
for (float i = 0; i < ls2; ++i)
{
pathtime2.at(i) = (lpath2.at(i)/0.2);
}


//model scintillation R/F and SiPM R/F
```

```
//models a random distribution after a rise and fall function
//then generate time shifts for summing function
cout << "calculating scintillation dist...\n";


for (float i = 0; i < scint.size(); ++i)
     {
tmp = sqroot->GetRandom();
ftime.at(i) = tmp + pathtime.at(i);
     }

for (float i = 0; i < scint2.size(); ++i)
     {
       tmp = sqroot->GetRandom();
ftime2.at(i) = tmp + pathtime2.at(i);
     }

cout << "done!\n";




//check for simultaneous hits on same pixel within "deadtime"
cout << "eliminating simultaneous hits...\n";
vector<float> temppos(3);
vector<float> temppos2(3);



int g = 0;
//each time a simulatenous event occurs the one with the highest time is deleted,
//ie the first that caused the discharge
//number of double hits are recorded and displayed
//one can unhighlight the cout's and confirm the correct ones are been deleted
for(int i = 0; i < 2; ++i)
{

int fpsize = finalpos.size();
int n =0;
bool simhit = true;
while(simhit)
{
temppos = finalpos.at(n);
p = 0;
bool simhit2 = true;
while(simhit2)
{ temppos2 = finalpos.at(p);
bool a, b, c;
```

```
a = (abs(temppos2.at(0) - temppos.at(0)) <= pixsize/2);
b = (abs(temppos2.at(1) - temppos.at(1)) <= pixsize/2);
c = (abs(ftime.at(n)-ftime.at(p)) <= deadtime);
if((n!=p) && a && b && c)
{

//cout<<"\n\nfirst\n"<<temppos.at(0)<<" & "<<temppos.at(1)<<" &
 "<<temppos.at(2)<<" & time:"<<ftime.at(n);
//cout<<"\n"<< temppos2.at(0)<<" & "<<temppos2.at(1)<<" &
 "<<temppos2.at(2)<<" & time:"<<ftime.at(p);
if(ftime.at(n) > ftime.at(p))
{
//cout <<"\ndel:\n"<<temppos.at(0)<<" & "<<temppos.at(1)<<" &
 "<<temppos.at(2)<<" & time:"<<ftime.at(n);
ftime.erase (ftime.begin() + n);
finalpos.erase (finalpos.begin() + n);
--n;
++g;
}


simhit2 = false;
}
if((p+1) == (fpsize -g))
simhit2 = false;
++p;
}
++n;
if(n == (fpsize -g) || (fpsize-g) == 1)
simhit = false;
}
}
cout << "\n done first";
cout << "\n" << finalpos2.size();
int g2 = 0;
for(int i = 0; i < 2; ++i)
{
int fpsize = finalpos2.size();
int n =0;
bool simhit = true;

while(simhit)
{
temppos = finalpos2.at(n);
p = 0;
bool simhit2 = true;
while(simhit2)
{ temppos2 = finalpos2.at(p);
bool a, b, c;
```

```
a = (abs(temppos2.at(0) - temppos.at(0)) <= pixsize/2);
b = (abs(temppos2.at(1) - temppos.at(1)) <= pixsize/2);
c = (abs(ftime2.at(n)-ftime2.at(p)) <= deadtime);
if((n!=p) && a && b && c)
{

//cout<<"\n\nfirst\n"<<temppos.at(0)<<" & "<<temppos.at(1)<<" &
 "<<temppos.at(2)<<" & time:"<<ftime2.at(n);
//cout<<"\n"<< temppos2.at(0)<<" & "<<temppos2.at(1)<<" &
 "<<temppos2.at(2)<<" & time:"<<ftime2.at(p);
if(ftime2.at(n) > ftime2.at(p))
{
//cout <<"\ndel:\n"<<temppos.at(0)<<" & "<<temppos.at(1)<<" &
 "<<temppos.at(2)<<" & time:"<<ftime2.at(n);
ftime2.erase (ftime2.begin() + n);
finalpos2.erase (finalpos2.begin() + n);
--n;
++g2;
}


simhit2 = false;
}
if((p+1) == (fpsize -g2))
simhit2 = false;
++p;
}
++n;
if(n == (fpsize -g2) || (fpsize-g2) == 1)
simhit = false;
}
}

cout << "done!\n";

//Final Signals
cout << "calculating Signal...\n";
int plotsize = 10000;
vector<float> graphs1A (plotsize), graphs1B (plotsize), graphs1C (plotsize), graphs1D
 (plotsize), graphs2A (plotsize), graphs2B (plotsize), graphs2C (plotsize), graphs2D
  (plotsize), gomp (3);
vector<int> Sfinpos1 (finalpos.size()), Sfinpos2 (finalpos2.size()) ;
float gx, gy, gr;
for(int i = 0; i < graphs1A.size(); ++i)
{
graphs1A.at(i) = -11.94*PEthres;
graphs1B.at(i) = -11.94*PEthres;
graphs1C.at(i) = -11.94*PEthres;
graphs1D.at(i) = -11.94*PEthres;
```

```
graphs2A.at(i) = -11.94*PEthres;
graphs2B.at(i) = -11.94*PEthres;
graphs2C.at(i) = -11.94*PEthres;
graphs2D.at(i) = -11.94*PEthres;
}
for(int i = 0; i < finalpos.size(); ++i)
{gomp = finalpos.at(i);
gx = gomp.at(0);
gy = gomp.at(1);
gr = sqrt(pow(gx,2)+pow(gy,2));
if(gy >= gx && gr >= (rad1+(rad2-rad1)/2))
Sfinpos1.at(i) = 1;
if(gy <= gx && gr >= (rad1+(rad2-rad1)/2))
Sfinpos1.at(i) = 2;
if(gy >= gx && gr <= (rad1+(rad2-rad1)/2))
Sfinpos1.at(i) = 3;
if(gy <= gx && gr <= (rad1+(rad2-rad1)/2))
Sfinpos1.at(i) = 4;
}
for(int i = 0; i < finalpos2.size(); ++i)
{gomp = finalpos2.at(i);
gx = gomp.at(0);
gy = gomp.at(1);
gr = sqrt(pow(gx,2)+pow(gy,2));
if(gy >= gx && gr >= (rad1+(rad2-rad1)/2))
Sfinpos2.at(i) = 1;
if(gy <= gx && gr >= (rad1+(rad2-rad1)/2))
Sfinpos2.at(i) = 2;
if(gy >= gx && gr <= (rad1+(rad2-rad1)/2))
Sfinpos2.at(i) = 3;
if(gy <= gx && gr <= (rad1+(rad2-rad1)/2))
Sfinpos2.at(i) = 4;
}

for(int n = 0; n < ftime.size(); ++n)
{
for(int i = 0; i < graphs1A.size(); ++i)
{
tomp = -(11.94)*(exp(-((i*.001)-ftime.at(n))/2.341)-exp(-((i*.001)
-ftime.at(n))/9.526));
if (tomp > 0)
{
if(Sfinpos1.at(n) == 1)
graphs1A.at(i) = tomp + graphs1A.at(i);
if(Sfinpos1.at(n) == 2)
graphs1B.at(i) = tomp + graphs1B.at(i);
if(Sfinpos1.at(n) == 3)
graphs1C.at(i) = tomp + graphs1C.at(i);
if(Sfinpos1.at(n) == 4)
```

```
graphs1D.at(i) = tomp + graphs1D.at(i);
}
else
{
if(Sfinpos1.at(n) == 1)
graphs1A.at(i) = 0 + graphs1A.at(i);
if(Sfinpos1.at(n) == 2)
graphs1B.at(i) = 0 + graphs1B.at(i);
if(Sfinpos1.at(n) == 3)
graphs1C.at(i) = 0 + graphs1C.at(i);
if(Sfinpos1.at(n) == 4)
graphs1D.at(i) = 0 + graphs1D.at(i);

}
}
}
//generate final signal (with geometry)
for(int n = 0; n < ftime2.size(); ++n)
{
for(int i = 0; i < graphs2A.size(); ++i)
{
tomp = -(11.94)*(exp(-((i*.001)-ftime2.at(n))/2.341)-exp(-((i*.001)
-ftime2.at(n))/9.526));
if (tomp > 0)
{
if(Sfinpos2.at(n) == 1)
graphs2A.at(i) = tomp + graphs2A.at(i);
if(Sfinpos2.at(n) == 2)
graphs2B.at(i) = tomp + graphs2B.at(i);
if(Sfinpos2.at(n) == 3)
graphs2C.at(i) = tomp + graphs2C.at(i);
if(Sfinpos2.at(n) == 4)
graphs2D.at(i) = tomp + graphs2D.at(i);
}
else
{
if(Sfinpos2.at(n) == 1)
graphs2A.at(i) = 0 + graphs2A.at(i);
if(Sfinpos2.at(n) == 2)
graphs2B.at(i) = 0 + graphs2B.at(i);
if(Sfinpos2.at(n) == 3)
graphs2C.at(i) = 0 + graphs2C.at(i);
if(Sfinpos2.at(n) == 4)
graphs2D.at(i) = 0 + graphs2D.at(i);

}
}
}
```

```
cout << "Done!\n";
vector<float> finalint;



//Filling and making Canvas

TCanvas *c1 = new TCanvas("c1","The FillRandom example",10,10,1000,700);
//TCanvas *c2 = new TCanvas("c2","The FillRandom example",150,10,700,700);
//TCanvas *c3 = new TCanvas("c3","The FillRandom example",200,10,700,700);
vector<float> tpos(3);
double x,y,z,xa,ya;
c1->cd();
   c1->SetFillColor(23);

TPad *pad1 = new TPad("pad1","The pad with the function",0.025,0.525,0.475,0.975,10);
TPad *pad2 = new TPad("pad2","The pad with the histogram",0.025,0.025,0.475,0.475,10);
TPad *pad3 = new TPad("pad3","The pad with the function",0.525,0.525,0.975,0.975,10);
TPad *pad4 = new TPad("pad4","The pad with the histogram",0.525,0.025,0.975,0.475,10);
pad1->Draw();
pad2->Draw();
pad3->Draw();
pad4->Draw();

//PAD1
//Histogram for arrival times
pad1->cd();
pad1->GetFrame()->SetFillColor(10);
pad1->GetFrame()->SetBorderMode(-1);
pad1->GetFrame()->SetBorderSize(5);
TH1D * hist = new TH1D("data", "Timing;Photon Arrival Time (ns); N", 100, -1, 40);
TH1D * hist2 = new TH1D("data2", "Timing;Photon Arrival Time (ns); N", 100, -1, 40);
hist->SetLineColor(1);
hist->SetFillStyle(3005);
hist->SetFillColor(4);
hist2->SetLineColor(1);
hist2->SetFillStyle(3005);
hist2->SetFillColor(2);

float timp, first1 = 100, first2 = 100;
for (float i = 0; i < ftime.size(); ++i)
{
    timp = ftime.at(i);
if(timp < first1)
first1 = timp;
        hist->Fill(timp);
}
for (float i = 0; i < ftime2.size(); ++i)
```

```
{
    timp = ftime2.at(i);
if(timp < first2)
first2 = timp;
        hist2->Fill(timp);
}
   hist->Draw();
hist2->Draw("same");
   c1->Update();

//PAD2
   pad2->cd();
   pad2->SetGridx();
   pad2->SetGridy();
   pad2->GetFrame()->SetFillColor(8);
   pad2->GetFrame()->SetBorderMode(-1);
   pad2->GetFrame()->SetBorderSize(5);

//Summing Graphs for final signal
TGraph *fsum = new TGraph();
for (float i = 0; i < graphs1A.size(); ++i)
{
x = i*.001;
tomp = graphs1A.at(i);
y = tomp;
fsum->SetPoint(i,x,y);
}
   fsum->SetLineColor(4);
   fsum->SetLineWidth(2);
//fsum->GetYaxis()->SetRangeUser(0,6000);
   fsum->GetXaxis()->SetRangeUser(0,100);
fsum->SetTitle("SiPM 1;Time (ns)");
   fsum->Draw("AL");
   c1->Update();
//w/ g
TGraph *f2sum = new TGraph();
for (float i = 0; i < graphs1B.size(); ++i)
{
x = i*.001;
tomp = graphs1B.at(i);
y = tomp;
f2sum->SetPoint(i,x,y);
}
   f2sum->SetLineColor(2);
   f2sum->SetLineWidth(2);
//f2sum->GetYaxis()->SetRangeUser(0,6000);
   f2sum->GetXaxis()->SetRangeUser(0,100);
   f2sum->Draw("SAME");
   c1->Update();
```

```
TGraph *f3sum = new TGraph();
for (float i = 0; i < graphs1C.size(); ++i)
{
x = i*.001;
tomp = graphs1C.at(i);
y = tomp;
f3sum->SetPoint(i,x,y);
}
   f3sum->SetLineColor(3);
   f3sum->SetLineWidth(2);
//f2sum->GetYaxis()->SetRangeUser(0,6000);
   f3sum->GetXaxis()->SetRangeUser(0,100);
   f3sum->Draw("SAME");
   c1->Update();
TGraph *f4sum = new TGraph();
for (float i = 0; i < graphs1D.size(); ++i)
{
x = i*.001;
tomp = graphs1D.at(i);
y = tomp;
f4sum->SetPoint(i,x,y);
}
   f4sum->SetLineColor(5);
   f4sum->SetLineWidth(2);
//f2sum->GetYaxis()->SetRangeUser(0,6000);
   f4sum->GetXaxis()->SetRangeUser(0,100);
   f4sum->Draw("SAME");
   c1->Update();

//intercepts
for(int i = 0; i < graphs1A.size(); ++i)
{
graphs1A.at(i) = abs(graphs1A.at(i));
graphs1B.at(i) = abs(graphs1B.at(i));
graphs1C.at(i) = abs(graphs1C.at(i));
graphs1D.at(i) = abs(graphs1D.at(i));
}

float lo = 100;
int loi = 0;
for(int i = 0; i < 9000; ++i)
{
tomp = graphs1A.at(i);
if(tomp < lo)
{lo = tomp;
loi = i;
}

}
```

```
float lo2 = 100;
float loi2 = 0;
for(int i = 0; i < 9000; ++i)
{
tomp = graphs1B.at(i);
if(tomp < lo2)
{lo2 = tomp;
loi2 = i;
}

}
float lo3 = 100;
int loi3 = 0;
for(int i = 0; i < 9000; ++i)
{
tomp = graphs1C.at(i);
if(tomp < lo3)
{lo3 = tomp;
loi3 = i;
}

}


float lo4 = 100;
float loi4 = 0;
for(int i = 0; i < 9000; ++i)
{
tomp = graphs1D.at(i);
if(tomp < lo4)
{lo4 = tomp;
loi4 = i;
}

}
if(lo < 0.2)
finalint.push_back(loi*.001);
if(lo2 < 0.2)
finalint.push_back(loi2*.001);
if(lo3 < 0.2)
finalint.push_back(loi3*.001);
if(lo4 < 0.2)
finalint.push_back(loi4*.001);
cout << "\n intercept1A: " << (loi*.001);

cout << "\n intercept1B: " << (loi2*.001);
```

```
cout << "\n intercept1C: " << (loi3*.001);

cout << "\n intercept1D: " << (loi4*.001);

//PAD3
pad3->cd();
   pad3->SetGridx();
   pad3->SetGridy();
   pad3->GetFrame()->SetFillColor(8);
   pad3->GetFrame()->SetBorderMode(-1);
   pad3->GetFrame()->SetBorderSize(5);

//Summing Graphs for final signal
TGraph *fsum2 = new TGraph();
for (float i = 0; i < graphs2A.size(); ++i)
{
x = i*.001;
tomp = graphs2A.at(i);
y = tomp;
fsum2->SetPoint(i,x,y);
}
   fsum2->SetLineColor(4);
   fsum2->SetLineWidth(2);
//fsum->GetYaxis()->SetRangeUser(0,6000);
   fsum2->GetXaxis()->SetRangeUser(0,100);
fsum2->SetTitle("SiPM 2;Time (ns)");
   fsum2->Draw("AL");
   c1->Update();
//w/ g
TGraph *f2sum2 = new TGraph();
for (float i = 0; i < graphs2B.size(); ++i)
{
x = i*.001;
tomp = graphs2B.at(i);
y = tomp;
f2sum2->SetPoint(i,x,y);
}
   f2sum2->SetLineColor(2);
   f2sum2->SetLineWidth(2);
//f2sum->GetYaxis()->SetRangeUser(0,6000);
   f2sum2->GetXaxis()->SetRangeUser(0,100);
   f2sum2->Draw("SAME");
   c1->Update();
TGraph *f3sum2 = new TGraph();
for (float i = 0; i < graphs2C.size(); ++i)
{
x = i*.001;
tomp = graphs2C.at(i);
y = tomp;
```

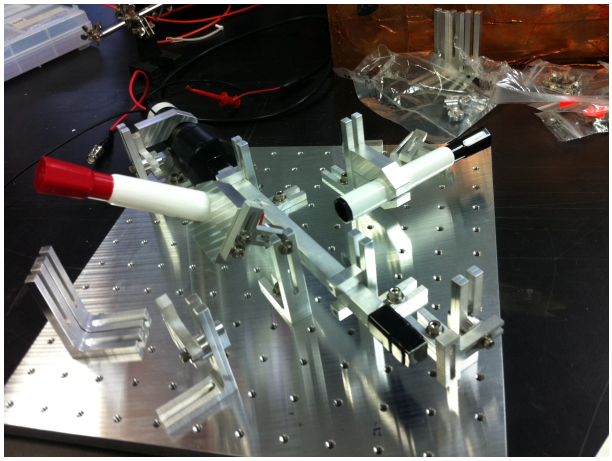
```
f3sum2->SetPoint(i,x,y);
}
   f3sum2->SetLineColor(3);
   f3sum2->SetLineWidth(2);
//f2sum->GetYaxis()->SetRangeUser(0,6000);
   f3sum2->GetXaxis()->SetRangeUser(0,100);
   f3sum2->Draw("SAME");
   c1->Update();
TGraph *f4sum2 = new TGraph();
for (float i = 0; i < graphs2D.size(); ++i)
{
x = i*.001;
tomp = graphs2D.at(i);
y = tomp;
f4sum2->SetPoint(i,x,y);
}
   f4sum2->SetLineColor(5);
   f4sum2->SetLineWidth(2);
//f2sum->GetYaxis()->SetRangeUser(0,6000);
   f4sum2->GetXaxis()->SetRangeUser(0,100);
   f4sum2->Draw("SAME");
   c1->Update();
c1->Update();
//intercepts
for(int i = 0; i < graphs1A.size(); ++i)
{
graphs2A.at(i) = abs(graphs2A.at(i));
graphs2B.at(i) = abs(graphs2B.at(i));
graphs2C.at(i) = abs(graphs2C.at(i));
graphs2D.at(i) = abs(graphs2D.at(i));
}

lo = 100;
loi = 0;
for(int i = 0; i < 9000; ++i)
{
tomp = graphs2A.at(i);
if(tomp < lo)
{lo = tomp;
loi = i;
}

}


lo2 = 100;
loi2 = 0;
for(int i = 0; i < 9000; ++i)
{
```

```
tomp = graphs2B.at(i);
if(tomp < lo2)
{lo2 = tomp;
loi2 = i;
}

}
lo3 = 100;
loi3 = 0;
for(int i = 0; i < 9000; ++i)
{
tomp = graphs2C.at(i);
if(tomp < lo3)
{lo3 = tomp;
loi3 = i;
}

}


lo4 = 100;
loi4 = 0;
for(int i = 0; i < 9000; ++i)
{
tomp = graphs2D.at(i);
if(tomp < lo4)
{lo4 = tomp;
loi4 = i;
}

}
if(lo < 0.2)
finalint.push_back(loi*.001);
if(lo2 < 0.2)
finalint.push_back(loi2*.001);
if(lo3 < 0.2)
finalint.push_back(loi3*.001);
if(lo4 < 0.2)
finalint.push_back(loi4*.001);
cout << "\n intercept2A: " << (loi*.001);

cout << "\n intercept2B: " << (loi2*.001);

cout << "\n intercept2C: " << (loi3*.001);

cout << "\n intercept2D: " << (loi4*.001);
//PAD4
pad4->cd();
   pad4->GetFrame()->SetFillColor(10);
```

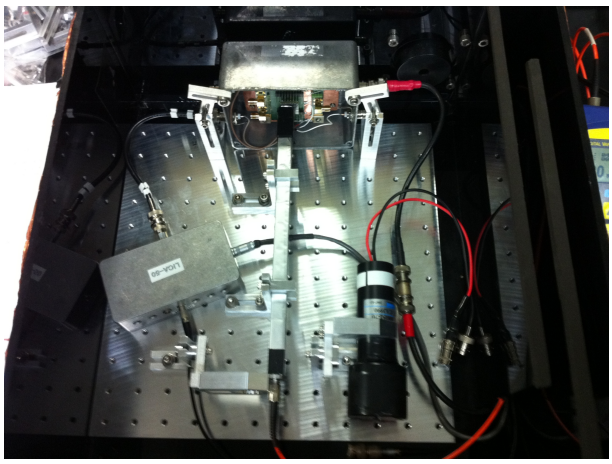
```
   pad4->GetFrame()->SetBorderMode(-1);
   pad4->GetFrame()->SetBorderSize(5);
TH1D * OutHist3 = new TH1D("data3", "Final Times", 5000, 0, 10);
for (float i = 0; i < finalint.size(); ++i)
{
    tomp = finalint.at(i);

        OutHist3->Fill(tomp);
}
OutHist3->SetLineColor(1);
OutHist3->SetFillStyle(3005);
OutHist3->SetFillColor(4);//blue
OutHist3->Draw();
 c1->Update();

//



//end of timing/signal generation
//display sim values

cout << "\n Simulatenous Pixel Hits1: " << g;
cout << "\n Simulatenous Pixel Hits1: " << g2;
cout << "\n Cross talk1:   " << xtalk;
cout << "\n Cross talk2:   " << xtalk2;
cout << "\n Detected Sensor Hits1:   " << ftime.size();
cout << "\n Detected Sensor Hits2:   " << ftime2.size();
cout << "\n First hit Time1: " << first1;
cout << "\n First hit Time2: " << first2;

cout << "\n Actual Displacement: " << stlength;
cout << "\n\n";




//Reset Mem
rand.clear();
lpath.clear();
pathtime.clear();
lpath2.clear();
pathtime2.clear();
NGtime.clear();
ftime.clear();
ftime2.clear();
scint.clear();
startpos.clear();
finalpos.clear();
```

```
finalpos2.clear();

vector<float>().swap(NGtime);
vector<float>().swap(rand);
vector<float>().swap(lpath);
vector<float>().swap(pathtime);
vector<float>().swap(ftime2);
vector<float>().swap(scint);
vector<float>().swap(ftime);


tmp = 0;



  return 0;
}
```

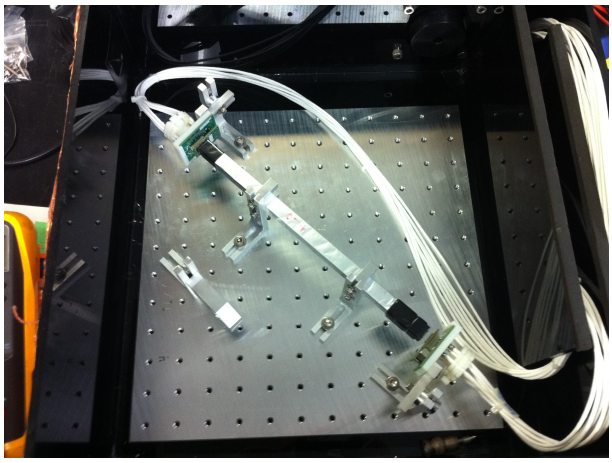
REFERENCES



(a)

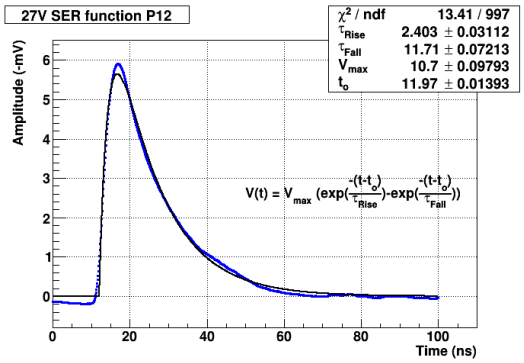


(b)

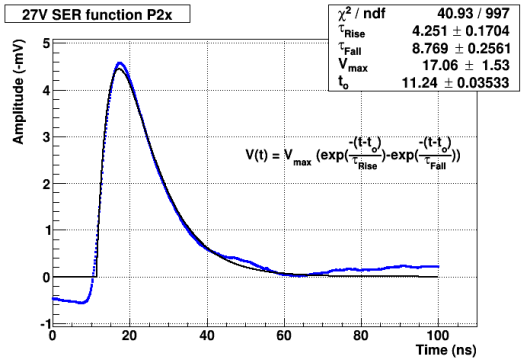


(c)

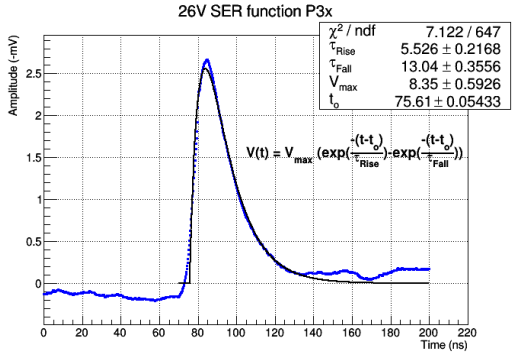
Figure 5.4: Displays the apparatus after fabrication and the many configurations the support pieces can take on the accomadate the parameters of detector tests



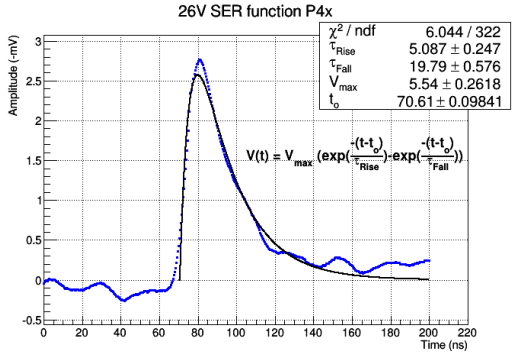
(a)



(b)

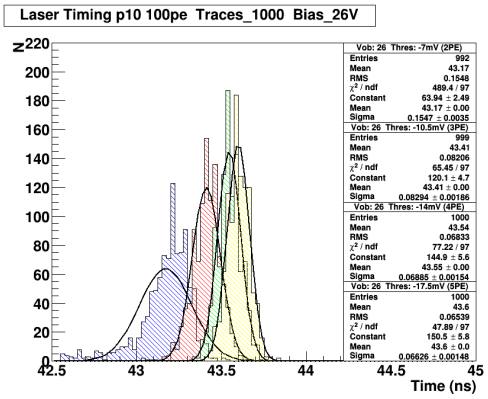


(c)

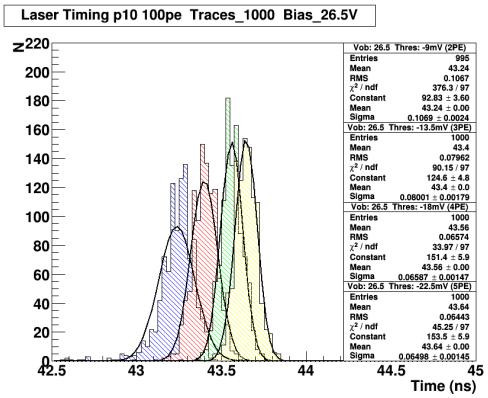


(d)

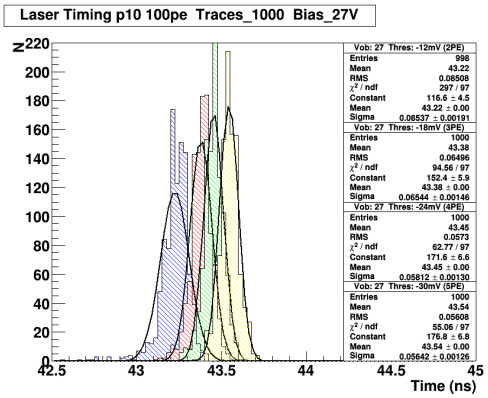
Figure 5.8: Displays the effect of multiplexing on rise and fall times. One should notice the artifact before the pulse becomes more significant for additional pixels and discredits the fit



(a)

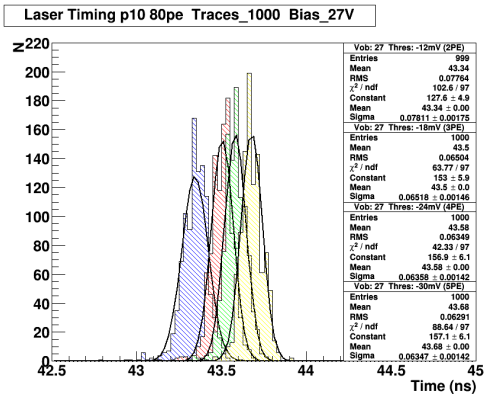


(b)

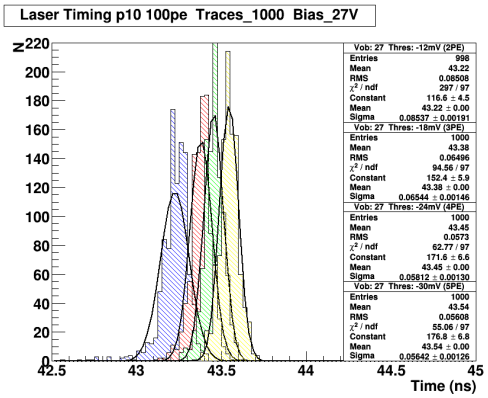


(c)

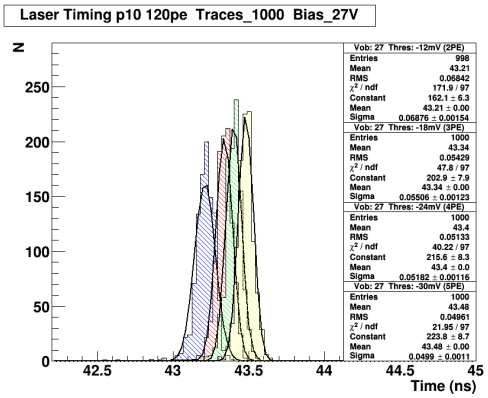
Figure 5.12: Displays the effect that applying different bias thresholds has on the timing of the detected signal



(a)



(b)



(c)

Figure 5.13: Displays the effect that applying different intensity laser levels has on the timing of the detected signal