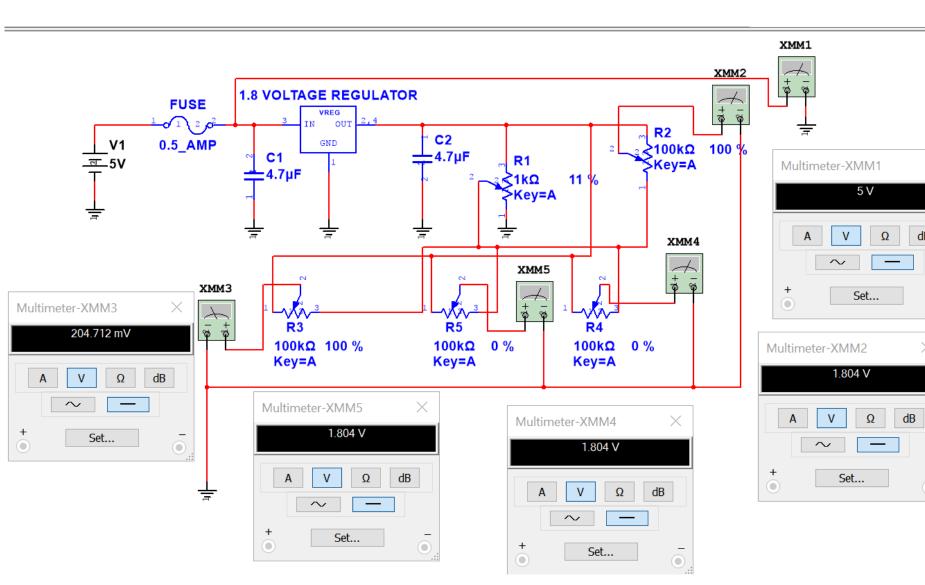


QCC Undergraduate Research Day Conference, Dec. 8<sup>th</sup>, 2017, Queensborough Community College

#### Building a power distribution unit (PDU)

A power distribution unit is an electrical circuit including a voltage divider that generates power to different electronic devices such as photomultiplier tubes (PMTs). The PDU contains a voltage divider that takes in 5V, drops 0.2mv – 0.25mv, and provides 1.8V; four variable potentiometers are used to provide 0.2V to 1.8V to power up to 4 PMTs. Each PMT module has an internal low voltage DC to high voltage DC converter which amplies the voltage 1V:1kV.

## **Results of Multisim simulation of pdu variable/regulated voltages**



The various multimeters shows voltage ouputs from different potentiometers.

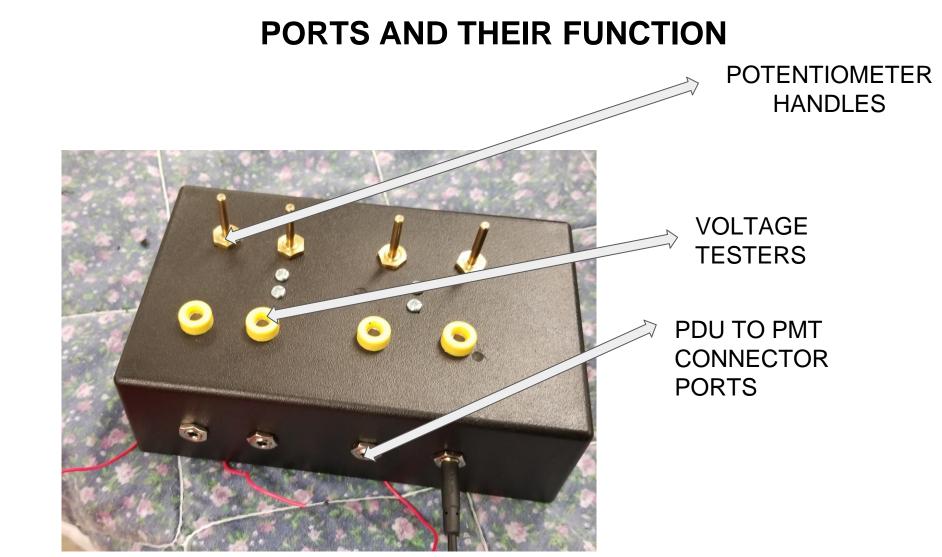
It shows the range of 0.2V to 1.8V, 5V from the power supply and ground can be measured by connecting a wire to the central ground of the entire circuit.

#### **MATH/WIRING DIAGRAM**

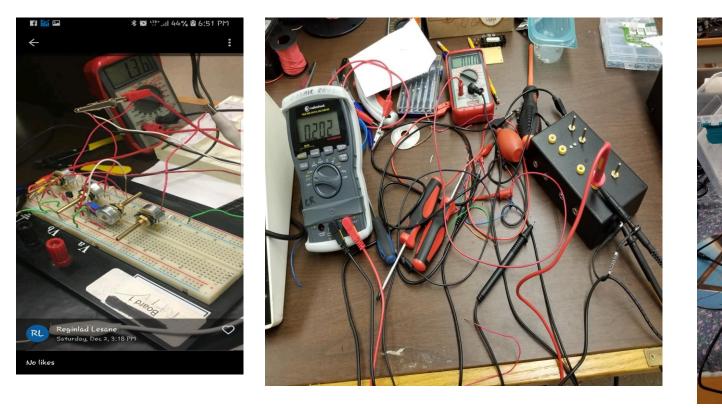
$$VOLTAGE RANGE = \left(\frac{R2}{R1 + R2} \times Vreg\right) - Vdeducted$$
$$\left(\frac{100k\Omega}{1k\Omega + 100k\Omega} \times 1.8V\right) - 0.2V = 0.6V$$

1.8V - 0.2V = 0.6V

The range voltage range here is calculated using the voltage divider rule (VDR) and basic math



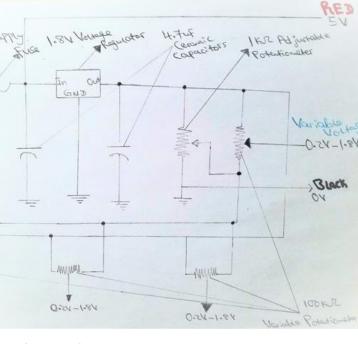
#### **TESTING THE PDU**



The PDU is being tested here between its highest range and lowest range. The lowest voltage shown here is displayed on the true RMS multimeter which is approximately what we needed. The value shown on the other multiplier is the peak voltage of the variable voltage; which is approximately 1.8V.







GND

•

LAWANEL\_\_\_

V

V8-1-1550

\*

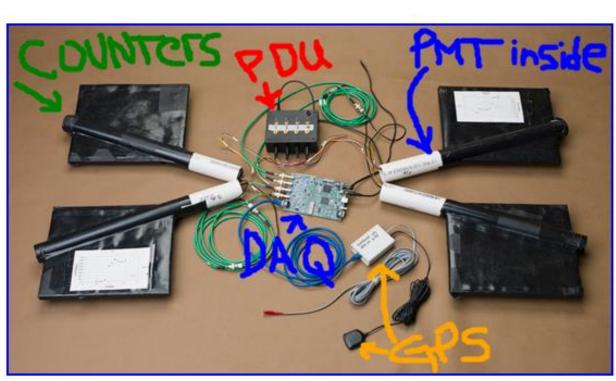


# **Working With Cosmic Rays**

Robert Li, Computer Engineering Technology, Queensborough Community College, Bayside, NY Jude Okocha, Electrical Engineering Technology, Queensborough Community College, Bayside, NY Reginald Lesane, Dept. of Physics, St. John's University, Jamaica, NY Raul Armendariz PhD, Dept. of Physics, Queensborough Community College, Bayside, NY Daniel Garbin PhD, Dept. of Math, Queensborough Community College, Bayside, NY

### Writing Python software to analyze Quark Net DAQ cosmic ray data

**Weight Reports:** The Fermilab Detector



- Our cosmic ray detectors use the QuarkNet DAQ board with hexadecimal data output.
- The DAQ reads a GPS satellite receiver to time stamp each particle detection with a UTC time to about a millisecond resolution.
- The DAQ data consists of up to 4 photomultiplier (PMT) outputs. • Each PMT output is pulse leading edge and trailing edge time of occurrence; the DAQ can distinguish time up
- to 1.25 nsec resolution.

#### Sample of EQUIP data recorded through the DAQ Board

= Leading Edge (A muon has entered a detector)

= Trailing Edge (A muon has left a detector)

The first column
a new event has
occurred
and returns an 80

nuon has lef	t a	ı de	ete	ctc	or)																
										<b>V</b>	7					· Day:	Mon	th·V	oor	Earr	mat
_																Day.		u i. i	eal	FUII	Παι
48450025	ВD	00	39	00	00	00	00	00	4738080B	012443	. 970	150917	v	01	0	+0033					
48468636																					
4867B29A	в7	00	32	00	00	00	00	00	4738080B	012443	.970	150917	V			+0033					
4867B29A														01	0	+0033					
4867B29B														01	0	+0033					
48D34B99	A1	00	20	00	00	00	00	00	48B5804B	012444	.978	150917	v	01	0	+0041					
48D34B99																					
4928DC75	AE	00	2D	00	00	00	00	00	48B5804B	012444	.978	150917	V	01	0	+0041					
4928DC75	00	00	00	35	00	00	00	00	48b5804b	012444	.978	150917	v	01	0	+0041					
4928DC75	00	3A	00	00	00	00	00	00	48B5804B	012444	.978	150917	V			+0041					
4949411F	Α3	00	21	00	00	00	00	00	48B5804B	012444	.978	150917	V	01	0	+0041					
494941 5	00	2C	00	2В	00	00	00	00	48B5804B	012444	.978	150917	v			+0041					
49DFEF18	80	00	27	00	00	00	00	00	48B5804B	012444	.978	150917	v	01	0	+0041					
49DFEF18	26	00	00	00	00	00	00	00	48B5804B	012444	.978	150917	V	01	0	+0041					
49DFEF18	00	34	00	35	00	00	00	00	48B5804B	012444	.978	150917	v	01	0	+0041					
49E927C5																					
49E927C6																					
4A458AA0	в7	00	00	00	00	00	00	00	4A32F88B	012445	.970	150917	v	01	0	+0033					
4A458AA0	00	00	38	00	00	00	00	00	4A32F88B	012445	.970	150917	v	01	0	+0033					
4A458AA1	00	2D	00	2A	00	00	00	00	4A32F88B	012445	.970	150917	V	01	0	+0033					
4A63026D	AD	00	2F	00	00	00	00	00	4A32F88B	012445	.970	150917	v	01	0	+0033					
4A63026D	00	3E	00	3C	00	00	00	00	4A32F88B	012445	.970	150917	v	01	0	+0033					
4AC2F405	в4	00	35	00	00	00	00	00	4A32F88B	012445	.970	150917	v	01	0	+0033					
4AC2F405	00	3D	00	3E	00	00	00	00	4A32F88B	012445	.970	150917	V	01	0	+0033					
4AD13A39														01	0	+0033					
4AD13A39														01	0	+0033					
4AD13A39	00	39	00	00	00	00	00	00	4A32F88B	012445	.970	150917	V	01	0	+0033					
4AE71C0C	в5	00	32	00	00	00	00	00	4A32F88B	012445	.970	150917	V								
4AE71C0C	00	3B	00	3E	00	00	00	00	4A32F88B	012445	.970	150917	۷	01	0	+0033					
			-						-												

### Old Program vs. Optimized Program

<pre>114 minutes = int((GoodToUse[x][0][TimeCheck][2:4])),</pre>	^	
<pre>.15 seconds = int((GoodToUse[x][0][TimeCheck][4:6])))))</pre>		
16		
17 c1 = 0.0803		
18 10 EventsDepInterval - []		
19 EventsPerInterval = [] 20		
.20 .21 #L5 = [datetime.timedelta(days = ds.days, seconds = ds.seconds//(5*60)*(5*60)) f	For ds in dtl #In minutes(ds) for ds in	
<pre>.22 print("Stage 5: Creating X, Y &amp; Z values for plot")</pre>	or us in all #[n_minales(us) for us in	
<pre>123 L5 = [datetime.timedelta(days = ds.days, seconds = ds.seconds//(5*60)*(5*60)) fo</pre>	or ds in dtl	
124		
<pre>125 for s, v in collections.Counter(L5).items():</pre>		
126 EventsPerInterval.append(v)		
.27		
128 CountsPerInterval = [(x/c1)/5 for x in EventsPerInterval] #Creates the number of	F counts within # time intervals	
129		
.30 del CountsPerInterval[-2:] #Gets rid of the last 2 values of pop1> (Adding '-	' will start from the end of the array!	
31		
<pre>132 diff = np.arange(0,(len(CountsPerInterval))+1) #Creates an array of ints with ev 132 advise = [wt5 for w in diff] #Creates a list with intervals of 5</pre>	ven spaces	
33 xAxis = [x*5 for x in diff] #Creates a list with intervals of 5 34		
34 35 del xAxis[-2:] #Removes the last 2 values of xAxis		
36 del CountsPerInterval[0] #Removes the first value of CountsPerInterval		
37		
38 zAxis = [x*100 for x in diff] #Creates intervals of 100 in an array		
39		
40		
41 #I am clueless on what these values represent		
<pre>42 yAxis = sum(1/x for x in CountsPerInterval)</pre>		
43 avgflux = sum(x for x in CountsPerInterval)		
44 yr = np.round(yAxis,4)		
45 avgf = yr/(len(CountsPerInterval))		
<pre>46 avgfper = [((((1/x)-avgf)/avgf)*100) for x in CountsPerInterval]</pre>		
47 avgfperr = np.round(avgfper,2)		
148 149 #del CountsPerInterval[0:2]		
150 #del xAxis[-2:]		
151 print("Stage 6: Setting up plot")		
<pre>ISI print( Stage 0. Setting up piet ) ISI NewWindow = plt.figure(1, figsize=[30,30]) #Creates a new window for plotting</pre>		
<pre>153 ax1 = NewWindow.add_subplot(111) #Creates subplot</pre>		
<pre>154 plt.scatter(xAxis,CountsPerInterval,color = "blue", label = "Flux")</pre>		
55 plt.xticks(zAxis)		
56 plt.yticks(color = "red")		
57 plt.locator_params(nbins = 4)		
<pre>8 plt.ticklabel_format(useOffset = False)</pre>		
<pre>9 plt.legend(bbox_to_anchor = (1,1), loc = 2, borderaxespad = 0.)</pre>		
50 plt.autoscale(enable = True, axis = "both", tight = None)		
<pre>il plt.xlabel("Time [Minutes]") il plt.xlabel("Wuen Flux Deviation (%)", color = "blue")</pre>		
52 plt.ylabel("Muon FLux Deviation (%)", color = "blue") 53 plt.title("Flux")		
64 ax2 = ax1.twinx()		
65 #plt.scatter(xAxis,bb, color = "red", label = "Pressure")		
$66 \ \#ptt.legend(bbox_to_anchor = (1,1), loc = 3, borderaxespad = 0.)$		
<pre>167 plt.ticklabel_format(useOffset = False)</pre>		
<pre>168 ax2.set_ylabel("Pressure Deviation(%)", color = "red")</pre>		
69 #plt.savefig("FluxPlot.png")		
70 plt.show()		

172 end = timer()
173 print("\nProcessing Complete!-----174 print("Seconds to Complete: "+str(end - start))

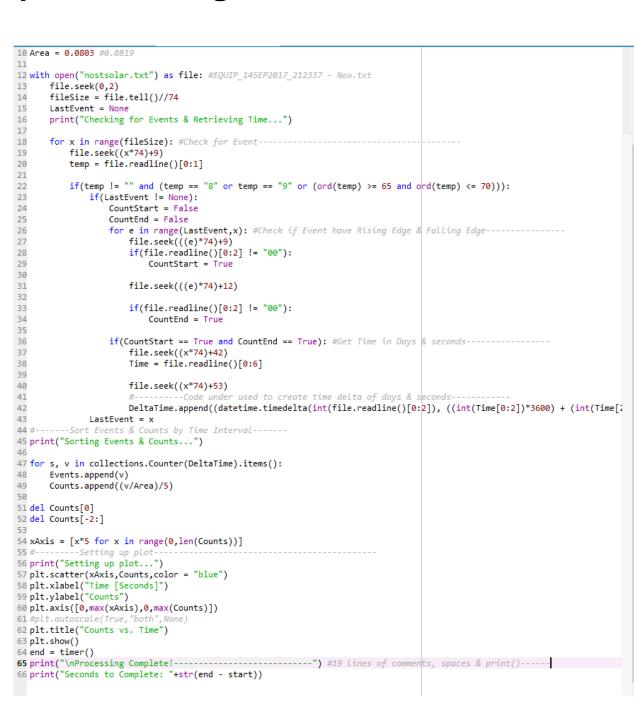
-This code is rewritten from a Version 1 pre-existing program written by my colleague David Buitrago to provide a better understanding of plotting the data.

-Version 1 can display other data such as atmospheric pressure and % Deviation.

-Currently the program will retrieve the # of counts over time by looking at the trailing edge and leading edge from the given file and produce a scatter plot in 5-minute intervals.

-The program takes time to generate a plot when loading text files into tables that average around 200MB or more. This is also due to some of the code that creates unnecessary lists with repeated data and take up memory.

-Using a 230MB Equip file, the average time it takes to generate a plot is around 30 minutes.



-With the help of Dr. D. Garbin, more efficient methods were implemented into the new Program with less than 50 lines of code compared To the previous program, which had 100-150 lines.

-Rather than storing all the data into multiple lists and taking up memory with repeated data, the idea is to only read and take specific columns of the file in one loop.

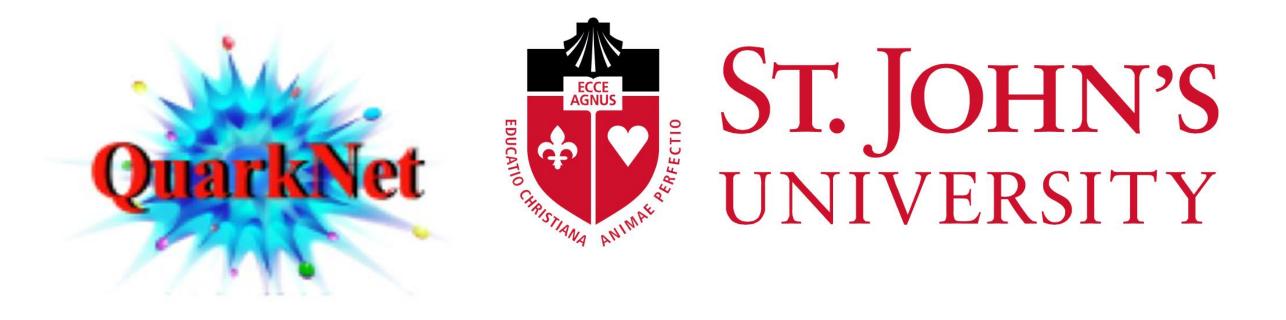
-Only three lists are declared and used – Counts, Events & Delta Time. Only one copy of each data is stored.

7 minutes.

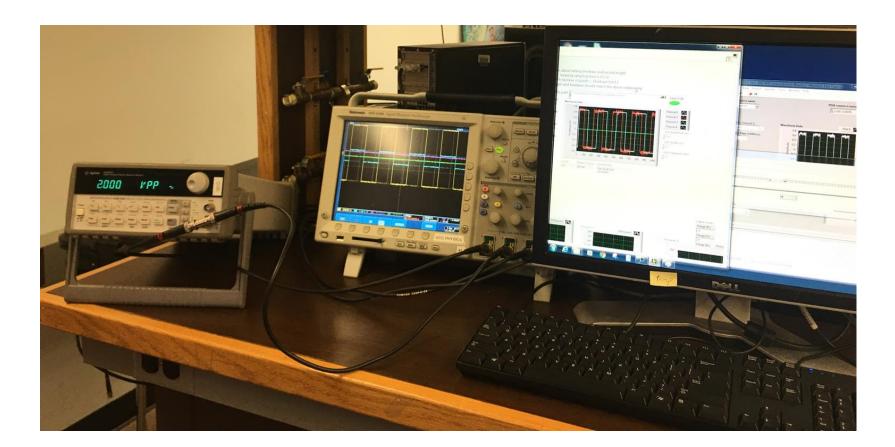
Hour:Minutes:Seconds Format in UTC Time

-Since the EQUIP file will have a fixed structure of columns and rows, the Seek() function will be used to reposition where the compiler will begin reading lines of data and skip redundant data, reducing memory usage.

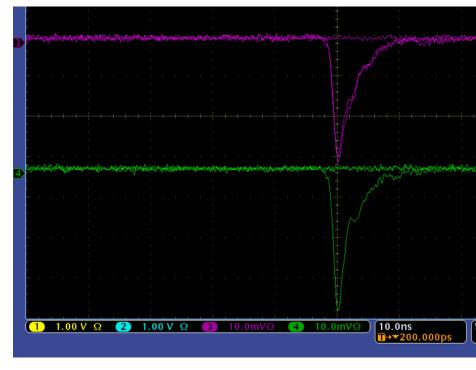
-Using the same 230MB file used in the previous Program, the average time to generate a plot is



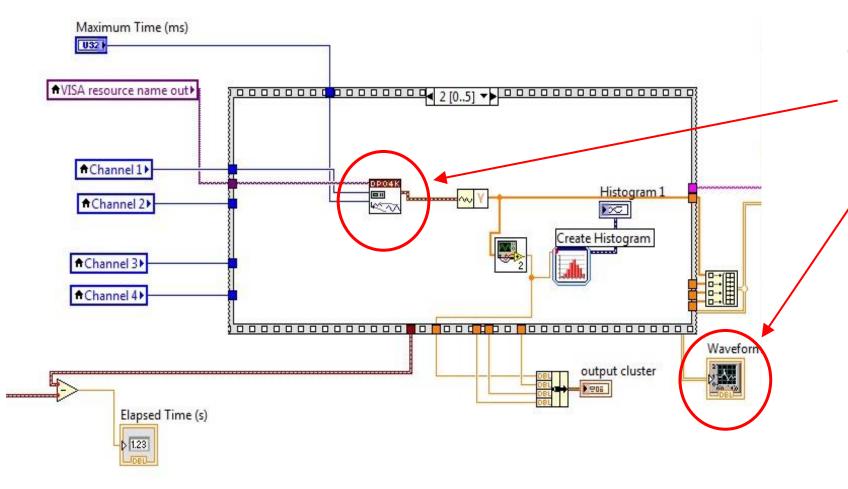
#### LabView as a Data Acquisition System from an Oscilloscope



We're using a Labview program to acquire signals from an oscilloscope to test photomultiplier tubes (PMTs). The LabView program written by Dr. Zhang from BNL retrieves signals at a 1Hz rate. We're trying to make this faster. To do this we are using a function generator to send a signal into a DPO4104 oscilloscope, and LabView is installed on a PC to control the oscilloscope and acquire the signals.

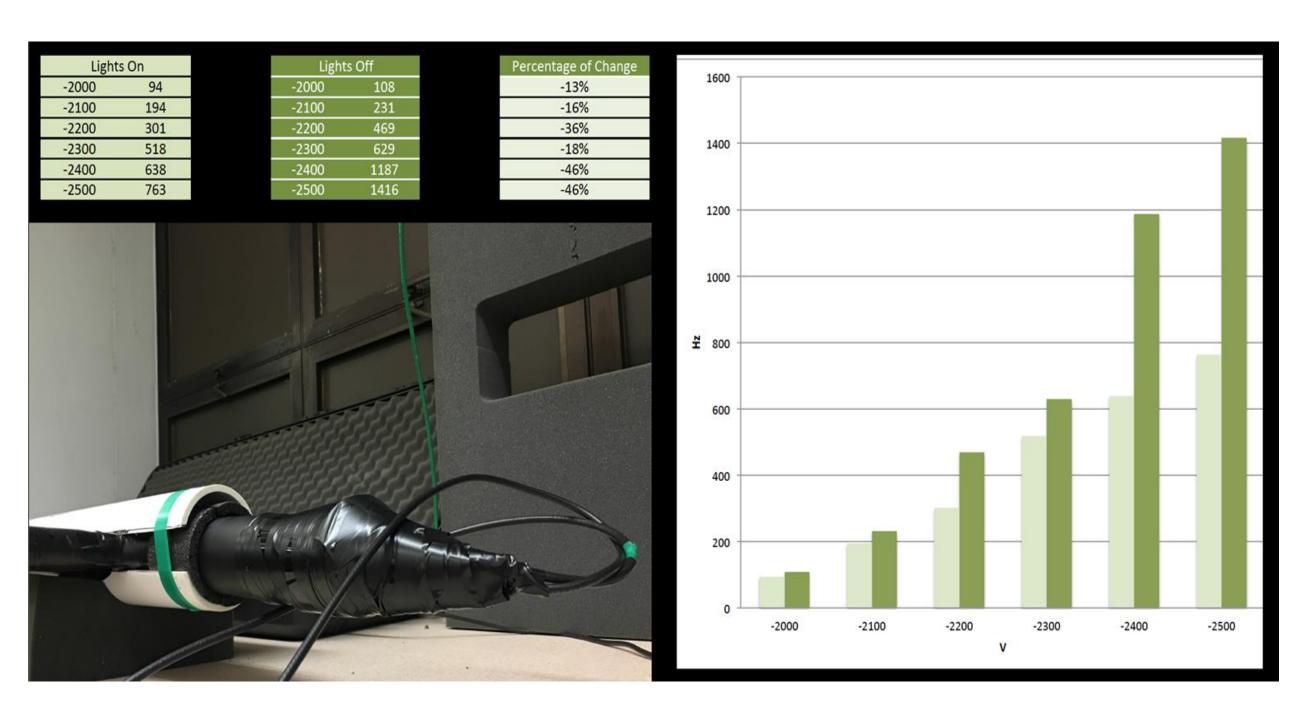


This is a DPO screenshot of two PMT noise signals coinciding >50ns. Both signals are captured in Run/Stop mode.



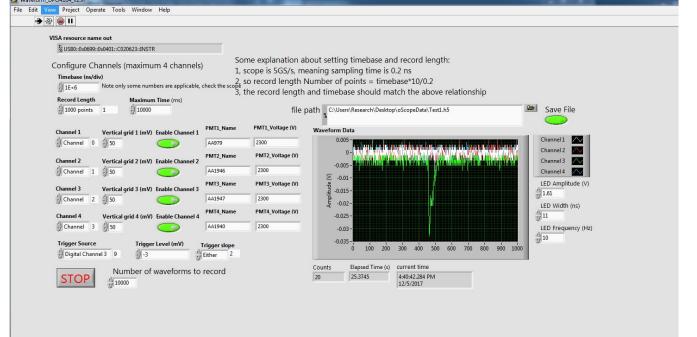
This is the LabView block diagram showing SubVI's, clusters, waveform charts and nodes. These components define how LabView communicates with the oscilloscope and downloads data. Also a National Instrument technician is assisting us.

### Building a Cosmic Ray Muon Detector and Sealing it from Ambient Light



A PMT and scintillator need to be wrapped light proof to detect reliable signals. A Lights On/ Lights Off test determine if light interferes with PMT noise signals. This is tested by turning room lights on and off to find the percentage of change in PMT noise signals. This chart has a small percent of change which indicates the equipment is light proof.

840-	84X	70049	ęĥ⇒γi	×k	€0₿4	Acul	670-48	w0.	d H	ار تام را ا	aleso	(metrika
ĝa.	ŝ	~\$~	مريد مريد	<b>1</b> 443)	e de la constante de la consta	lage/h	dym.	ownika	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	iyinni	140	<u>Arya</u> r
												+
			ł					-				
00	G	s/s boi	nte		>	< >	H	Η	£	$\rightarrow$		



This is a LabView screenshot of two PMT noise signals coinciding >50ns. However the signals are captured in Single mode which displays.

We are going to modify the DPO SubVI to see if it will acquire more waveforms per second.