Muon Physics

Paul Lea and Alex Weiss. Section 1

The primary research objective of this lab was to produce an accurate measurement for muon lifetimes on earth's surface. Secondary objectives included verrifiyng the lab's equipment for this project through testing frequency response, saturation voltage, FPGA response time and discriminator functionality. Additionally, we used the derived muon lifetimes to derive the Fermi coupling constant as well as the ratio of positive to negative muons.

Section 1: Frequency Response

The objective of this section was to verrify the frequency response of the amplifier over the range of frequencies we will need for this experiment.

Equipment:

- Muon physics electronics box
- Oscilliscope

Procedure

- Applied various frequency 100mV peak-to-peak sine wave from a Function Generator to the PMT Input of the electronics box.

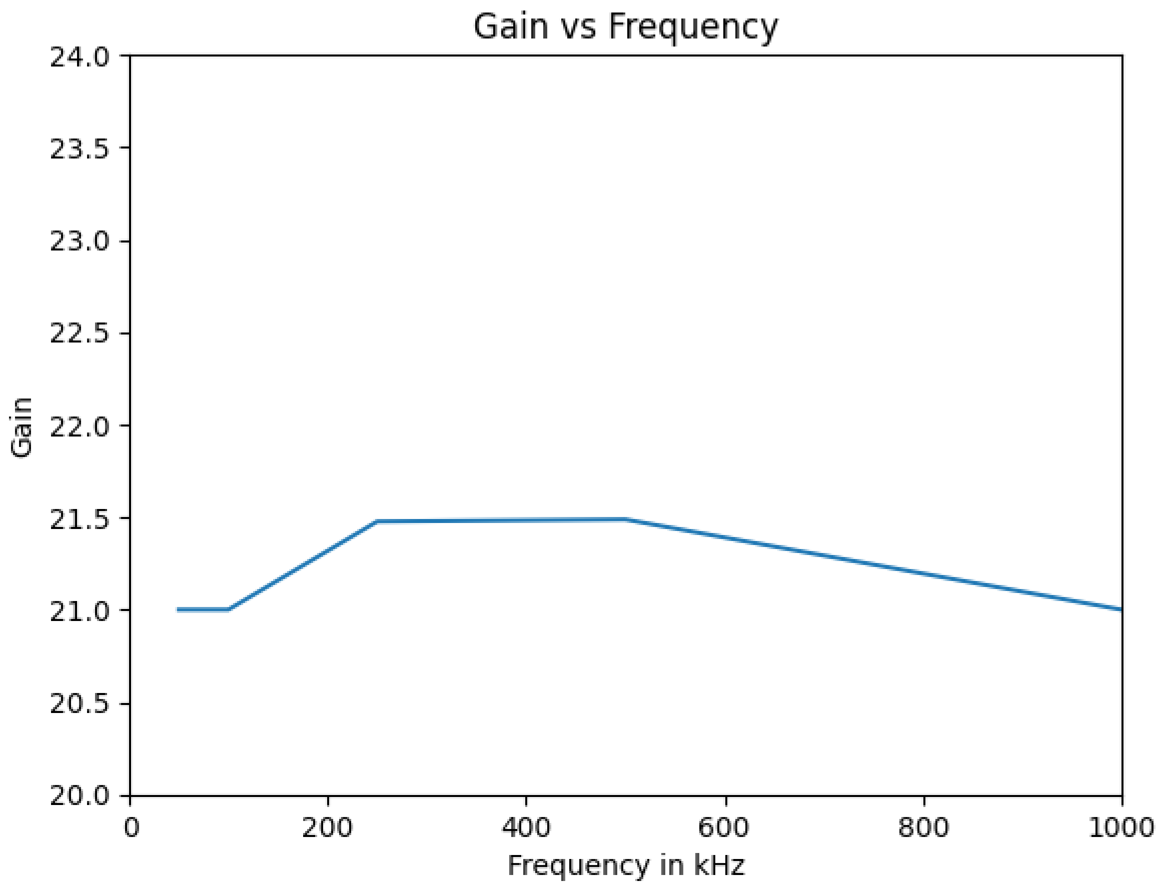
- Measured the Amplifier Output and calculated the ratio Vout/Vin, representing the amplifier's gain, for each frequency tested.

- Accounted for the attenuation resistors between the amplifier output and the front panel connector by multiplying this ratio by 21 (1050/50), considering a 1% uncertainty in this factor, to determine the actual amplifier gain.

Data

In [37]:
```
#Import Statements
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import matrepr as mpr
%load_ext matrepr
from scipy.optimize import curve_fit
from scipy.odr import Model, RealData, ODR
import warnings
warnings.filterwarnings('ignore')

vin = np.array([88,88,90,88,88])
vout = np.array([88,88,88,86,88])
gain = vin/vout * 21
freq = 50,100,250,500,1000
plt.axis([0, 1000, 20,24])
plt.xlabel("Frequency in kHz")
plt.ylabel("Gain")
plt.title("Gain vs Frequency")
plt.plot(freq, gain)
plt.show()
```

```
The matrepr extension is already loaded. To reload it, use:
  %reload_ext matrepr
```



> Figure 1: Plot of Gain vs Frequency
>
> ## Analysis:

The frequency response for this amplifier circuit is very flat. Throughout the 50kHz-1mHz range, the response remains identical. It is hard to estimate the exact maximum decay rate, but it is likely well in excess of 1 megahertz, or much less then 1 microsecond.

Conclusions:

The frequency response of the amplifier is very flat. There is a constant and steady gain of 1 throughout the 50kHz to 1mHz regime. This is expected and beneficial to our experiements occuring later on.

Section 2: Saturation output voltage

The objective of this portion of the lab was to determine the saturation voltage for our amplifier used in this exeperiment.

Equipment

- Oscilliscope

- Muon electronics box

Procedure

- Applied various voltage 100kHz sine waves from a Function Generator to the PMT Input of the electronics box.
- Took measurements of each peak-peak output voltage for each input voltage
- Noted any falloff points or discrepancies

Data

In [38]:
```
vin = np.array([105,148,202, 250, 300])
vout = np.array([104,138,168,190,214])
vinerr = vin * 0.03
vouterr = vout * 0.03
plt.title("Output voltage as a function of input voltage")
plt.xlabel("Input voltage (mv)")
plt.ylabel("Output voltage")
plt.errorbar(vin,vout, xerr =vinerr, yerr = vouterr)
plt.show()
```

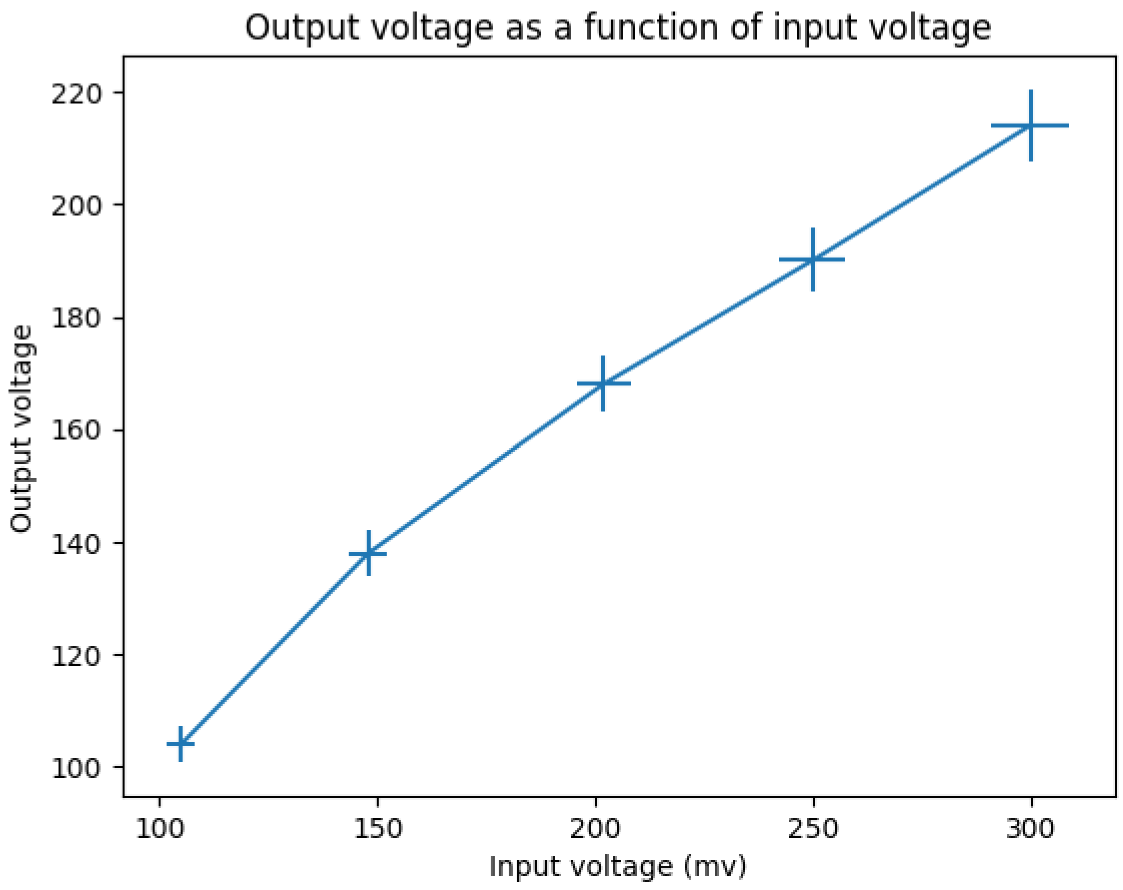
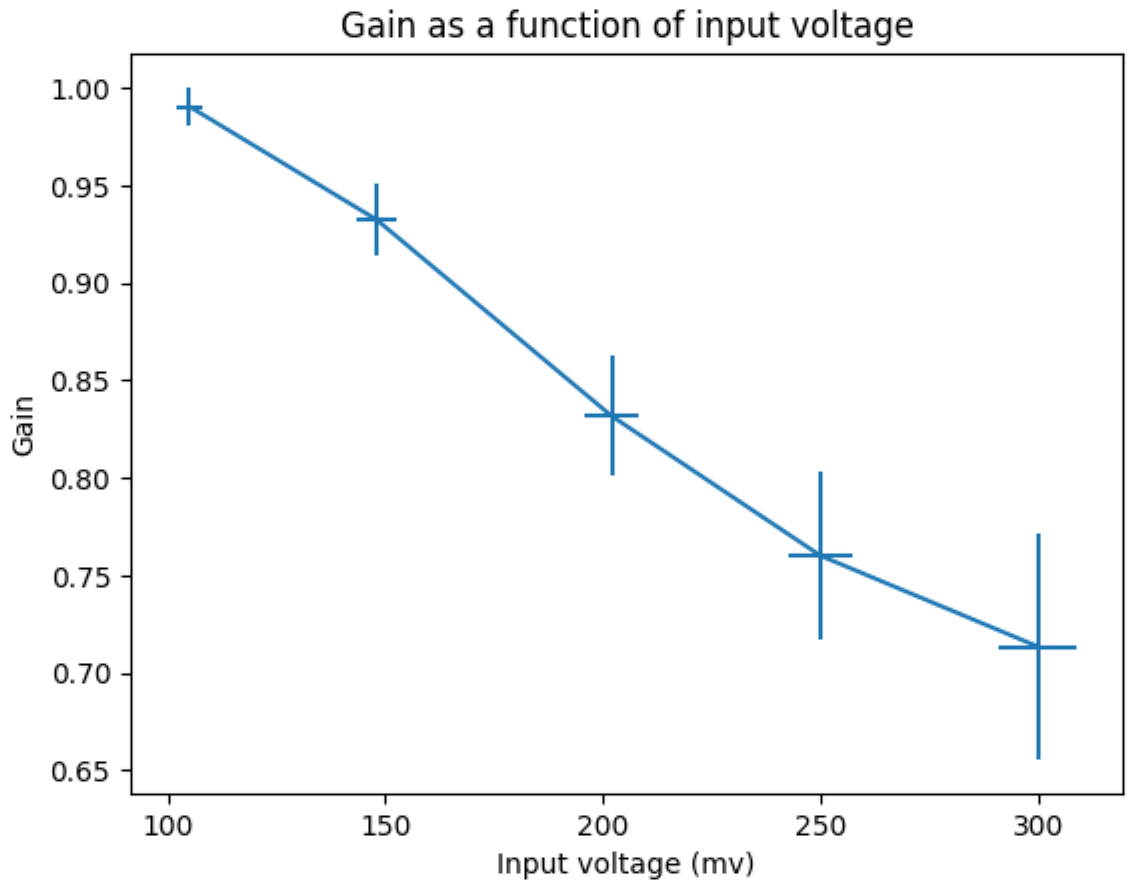


Figure 2: Plot of output voltage as a function of input voltage

In [39]:
```
gain = vout/vin
gainerr = vouterr * vinerr / 1000
plt.errorbar(vin, gain, xerr = vinerr, yerr = gainerr )
plt.title("Gain as a function of input voltage")
plt.xlabel("Input voltage (mv)")
plt.ylabel("Gain")
plt.show()
```



> Figure 3: Gain as a function of input voltage
>
> ### Analysis:
>
> Voltage saturation began to occur above 100mv. The effect was slight though, and the amplifier continued to perform moderately well past its saturation point. The FPGA did not appear to be affected, and the waveforms appeared to qualtiatively be in sync.

Conclusions

The amplifier performed as expected, with voltage gain dropoff occuring above 100mv. The gain calculated resulted in larger errors towards the higher end of the voltage spectrum due to error propagation between input voltage and output voltage.

Section 3: Discriminator

The objective of this section was to examine the behavior of the discriminator function of the Muon Physics electronics setup. This was acomplished by adjusting the threshold value and observing the changes in the time difference.

Equipment

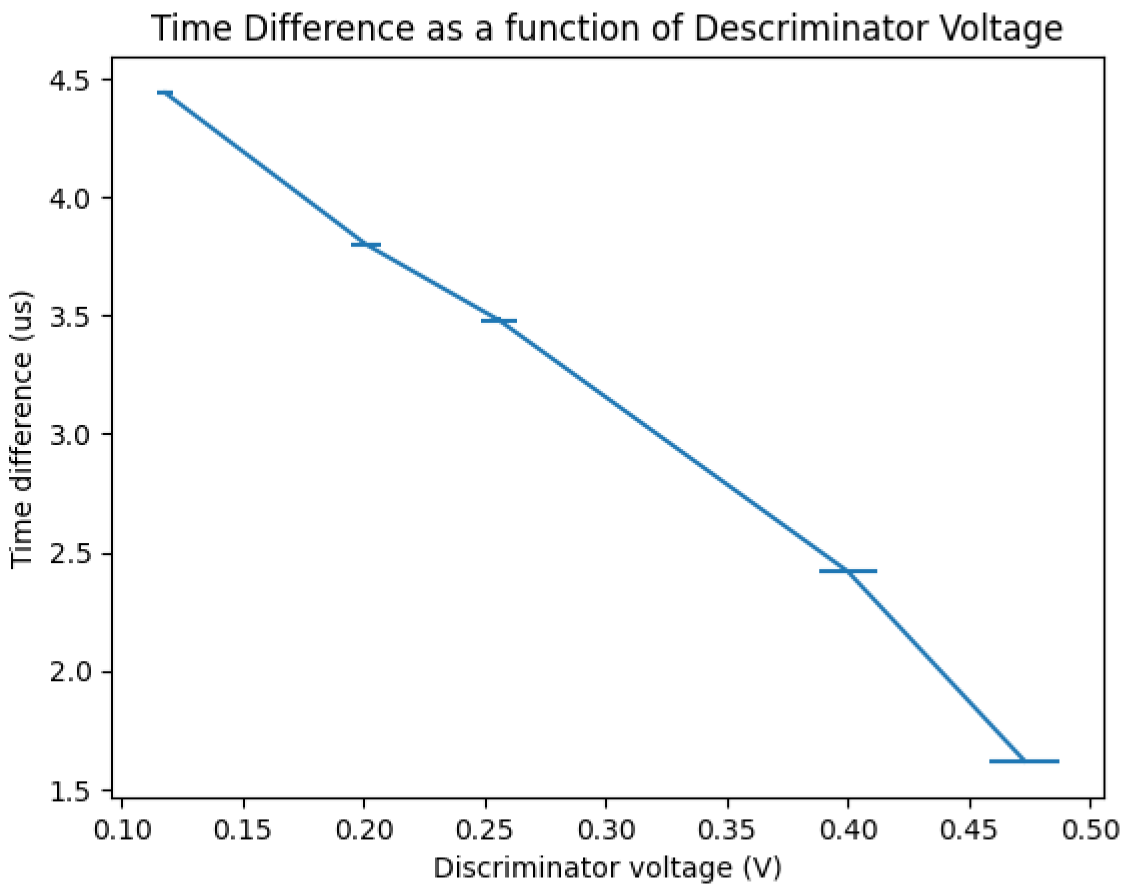
- Oscilliscope
- Muon electronics box
- Digital Multimeter

Procedure

- Applied a 100kHz 100mV sine signal into the PMT input of the elctronics box.
- Adjusted the descriminator threshold to various levels and observed the behavior recoreded on oscilliscope.
- Used oscilliscope to measure time interval and voltage max using cursor search feature.
- Repeated for every discriminator value

Data

In [40]:
```
discrimV = np.array([0.118,0.201,0.256, 0.400,0.473])
deltaT = np.array([4.440,3.800,3.480, 2.420,1.620])
discrimErr = discrimV * 0.03
deltaTerr = np.array([0.01,0.01,0.01,0.01,0.01])

plt.errorbar(discrimV, deltaT, xerr = discrimErr, yerr = deltaTerr)
plt.title("Time Difference as a function of Descriminator Voltage")
plt.xlabel("Discriminator voltage (V)")
plt.ylabel("Time difference (us)")
plt.show()
```



> Figure 4: Time difference as a function of Descriminator Voltage

In [41]:
```
ampVoltage = np.array([56,58,68,104,124,])
ampVoltErr = ampVoltage * 0.03
plt.errorbar(discrimV, ampVoltage, xerr = discrimErr, yerr = ampVoltErr)
plt.title("Amplifier voltage on rising edge as a function of Discriminator voltage ")
plt.ylabel("Amplifier voltage (mv)")
plt.xlabel("Discriminator Voltage (V)")
plt.show()
```

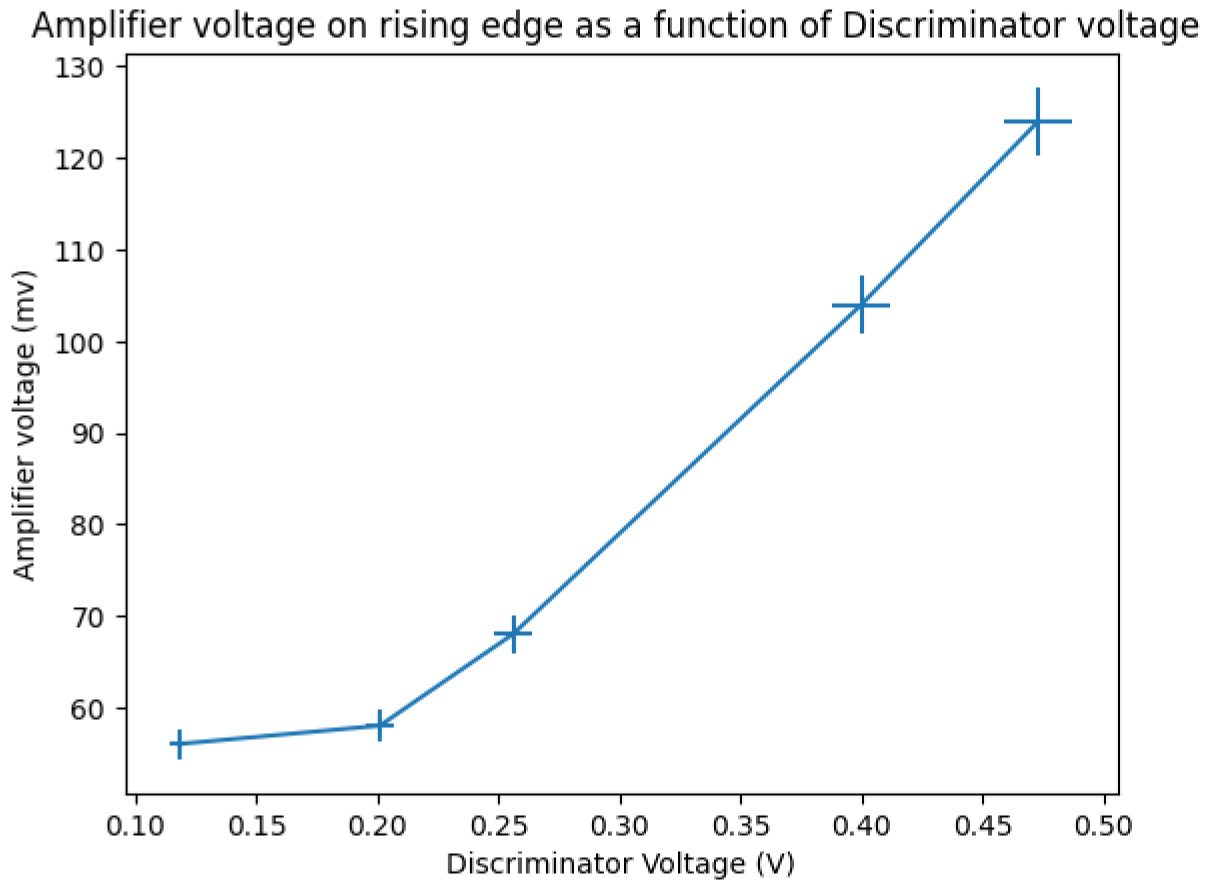


Figure 5 Amplifier voltage on rising edge as a function of discriminator voltage

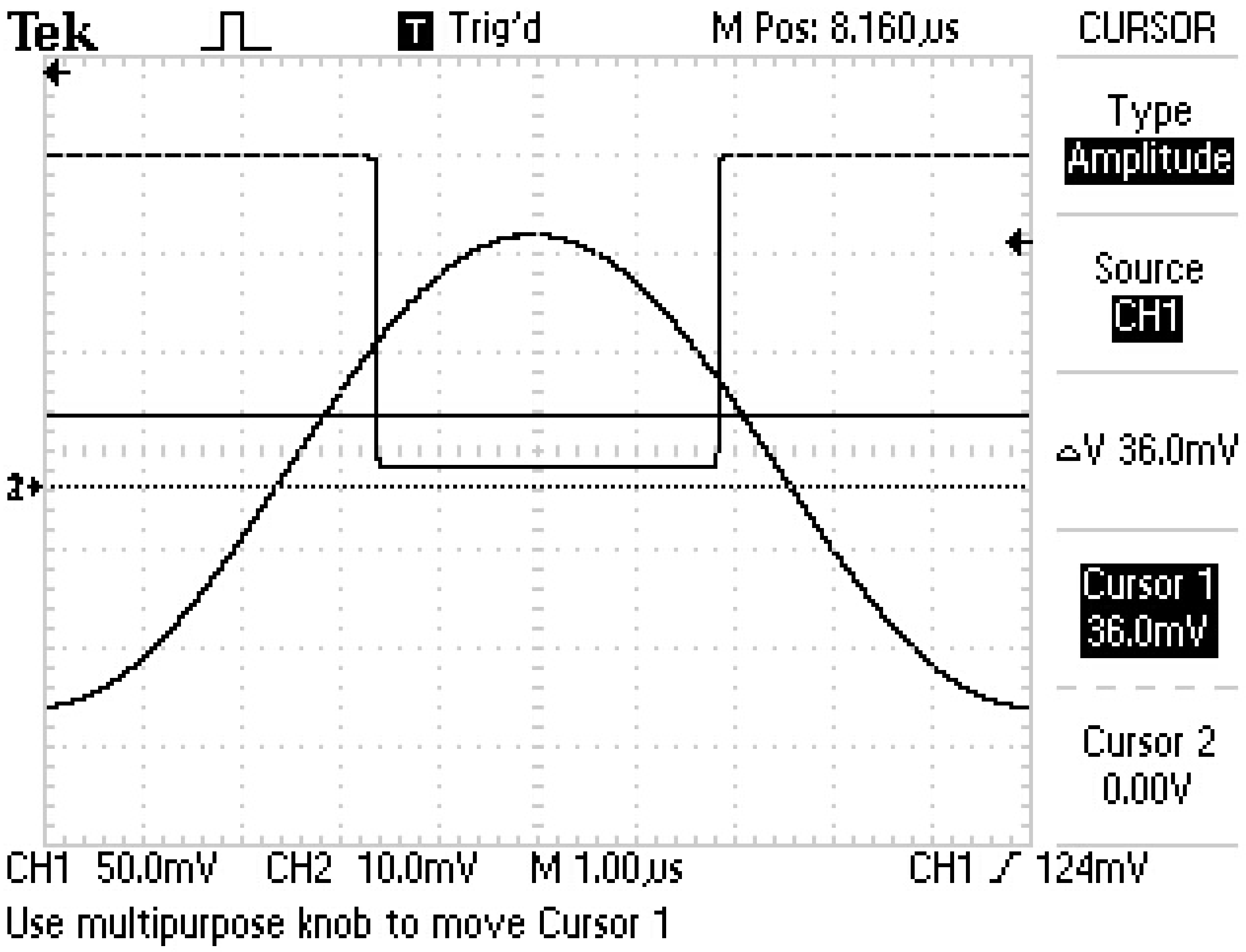


Figure 6: Single discriminator and amplifier signal waveform

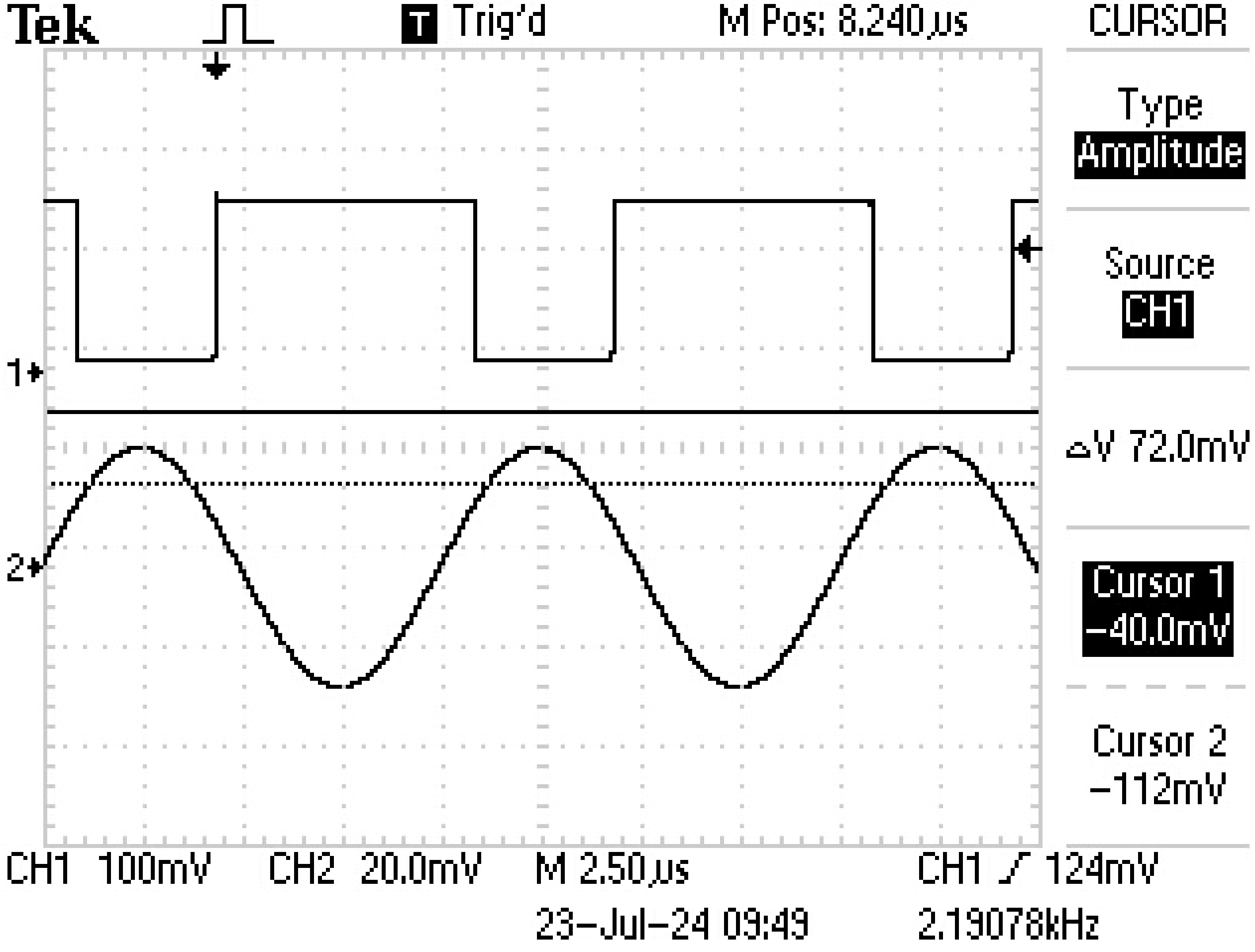


Figure 7: Several discriminator and amplifier signal waveforms

Analysis:

This section detailed the usage of the descriminator, and its sensitivity to voltage fluctuation. By setting the threshold at any level desired, it allows an arbitrary waveform to be converted into a series of pulses.

Conclusion:

The descriminator functioned as intended. The waveforms qualitatively looked good and had sharp rising and falling edges. The time difference and amplifier voltage were roughly linear, and error in those may be due to the inpercision of the oscilliscope cursors.

Section 4: Timing properties of FPGA

The objective of this section was to verrify the properties of the FPGA communication interface with the PC used to record muon events later on. Importantly, we determined the maximum and minimum event times observable by the computer.

Equipment

- Oscilliscope
- Muon electronics box
- Scintillator (and LED pulser)
- Desktop with Muon software

- Digital Multimeter

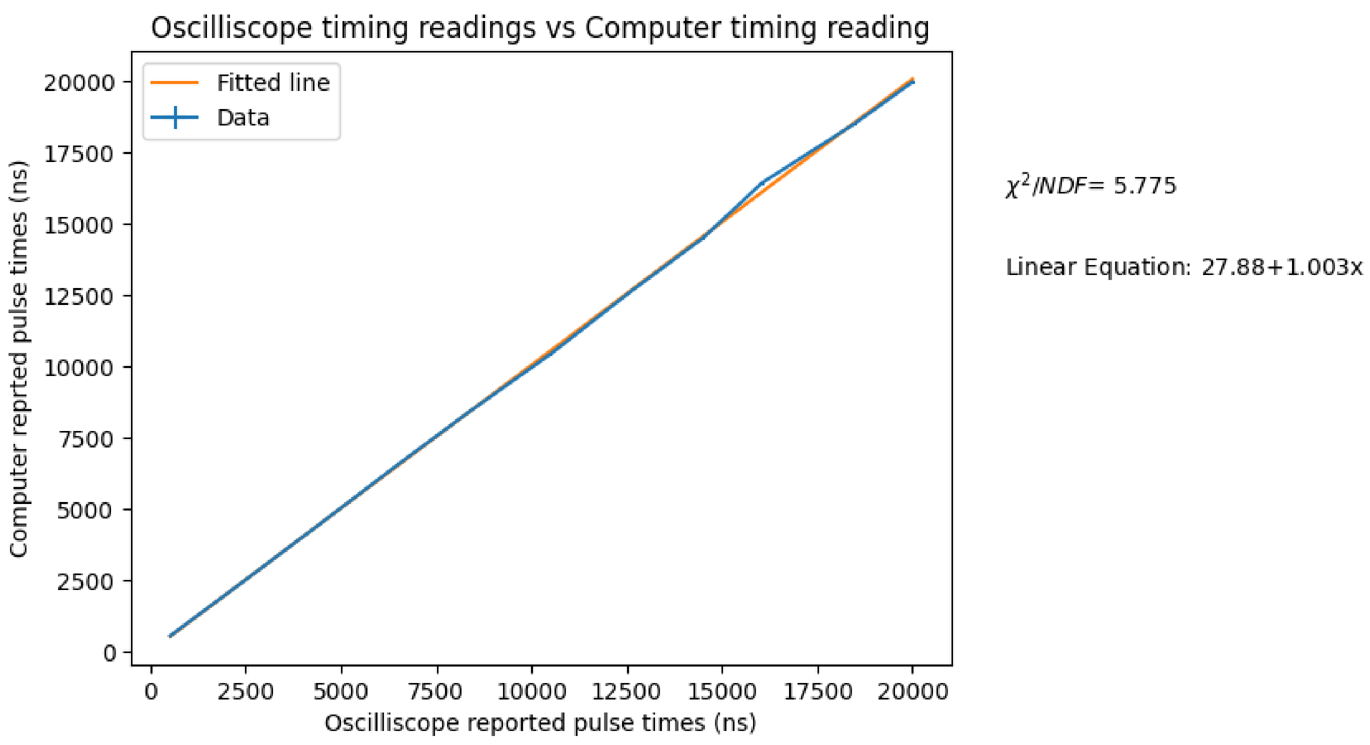
Procedure

- We switched from using the function generator to using the pulse input of the scintillator tube.
- Measured the time between successive pulses using the cursors on the oscilliscope.
- Next, we repeated this same measurement using the computer interface.
- These measurements were repeated for various values of pulse length, from 0.5us to 20us.

Data

In [42]:
```
t1 = pd.read_csv("AlexPaul/Section 4/0.5microsec reading (trial 1).txt", sep="\t").iloc[
t2 = pd.read_csv("AlexPaul/Section 4/2.5microsec reading (trial 2).txt", sep="\t").iloc[
t3 = pd.read_csv("AlexPaul/Section 4/4.5microsec reading (trial 3).txt", sep="\t").iloc[
t4 = pd.read_csv("AlexPaul/Section 4/6.5microsec reading (trial 4).txt", sep="\t").iloc[
t5 = pd.read_csv("AlexPaul/Section 4/8.5microsec reading (trial 5).txt", sep="\t").iloc[
t6 = pd.read_csv("AlexPaul/Section 4/10.5microsec reading (trial 6).txt", sep="\t").iloc
t7 = pd.read_csv("AlexPaul/Section 4/12.5microsec reading (trial 7).txt", sep="\t").iloc
t8 = pd.read_csv("AlexPaul/Section 4/14.5microsec reading (trial 8).txt", sep="\t").iloc
t9 = pd.read_csv("AlexPaul/Section 4/16.5microsec reading (trial 9).txt", sep="\t").iloc
t10= pd.read_csv("AlexPaul/Section 4/18.5microsec reading (trial 10).txt", sep="\t").ilo
t11= pd.read_csv("AlexPaul/Section 4/20.0microsec reading (trial 11).txt", sep="\t").ilo

t1avg = np.average([t1[t1 <= 30000]])
t2avg = np.average([t2[t2 <= 30000]])
t3avg = np.average([t3[t3 <= 30000]])
t4avg = np.average([t4[t4 <= 30000]])
t5avg = np.average([t5[t5 <= 30000]])
t6avg = np.average([t6[t6 <= 30000]])
t7avg = np.average([t7[t7 <= 30000]])
t8avg = np.average([t8[t8 <= 30000]])
t9avg = np.average([t9[t9 <= 30000]])
t10avg = np.average([t10[t10 <= 30000]])
t11avg = np.average([t11[t11 <= 30000]])

t1std = np.std([t1[t1 <= 30000]])
t2std = np.std([t2[t2 <= 30000]])
t3std = np.std([t3[t3 <= 30000]])
t4std = np.std([t4[t4 <= 30000]])
t5std = np.std([t5[t5 <= 30000]])
t6std = np.std([t6[t6 <= 30000]])
t7std = np.std([t7[t7 <= 30000]])
t8std = np.std([t8[t8 <= 30000]])
t9std = np.std([t9[t9 <= 30000]])
t10std = np.std([t10[t10 <= 30000]])
t11std = np.std([t11[t11 <= 30000]])

oscilliscopeReadings = np.array([0.52,2.48,4.48,6.52,8.40,10.50,12.60,14.50,16.05,18.50,
oscilliscopeReadings = oscilliscopeReadings * 1000 # convert to ns
computerReadings = np.array([t1avg, t2avg, t3avg, t4avg, t5avg, t6avg,t7avg,t8avg,t9avg,
computerErr = np.array([t1std,t2std,t3std, t4std, t5std,t6std, t7std, t8std, t9std,t10st
oscilliscopeErr = np.ones(11) * 0.05 * 1000
def lin(params,x):
    return params[0] + params[1] * x
model = Model(lin)
odrData       = RealData(oscilliscopeReadings, computerReadings ,sx = oscilliscopeErr, s
params        = 27,1
odr           = ODR(odrData, model, beta0=params)
odr_result    = odr.run()
params_fit1 = odr_result.beta
covar_fit1  = odr_result.sd_beta
chisquared1 = odr_result.res_var
```

```
V_fit = lin(params_fit1,oscilliscopeReadings)
plt.errorbar(oscilliscopeReadings, computerReadings, xerr= oscilliscopeErr, yerr = compu
plt.plot(oscilliscopeReadings, V_fit, label="Fitted line")
plt.title("Oscilliscope timing readings vs Computer timing reading")
plt.ylabel("Computer reprted pulse times (ns)")
plt.xlabel("Oscilliscope reported pulse times (ns)")
plt.gcf().text(0.95,0.7, "$\\chi^2/NDF$= " + str(np.round(chisquared1,3)))
plt.gcf().text(0.95,0.6, "Linear Equation: " + str(np.round(params_fit1[0],3)) + "$ + $"
plt.legend()
plt.show()
```



> Figure 8: Oscilliscope timings vs Computer timings with a linear curve fit

All points greater then 30us were trimmed out to remove sources of interference. The detector appears to reach a timeout at 40us.

Analysis:

The maximum event time appears to be 40100 ns, or 40.1 us. This would corrospond to the timeout time mentioned in the literature. The minimum event time is shorter then the first data point 0.5us, as linearity was preserved throughout the range of pulse times tested.

Conclusions:

The FPGA's response range was satisfactory. All muon events relayed to the FPGA should be able to be recorded via the computer interface. The response compared to the oscilliscope is highly linear, with a slope of 1.003. The error of 0.003 was more likely due to oscilliscope cursor error, as it was difficult to align the oscilliscope cursors at the exact same point each time.

Section 5:Muon Physics

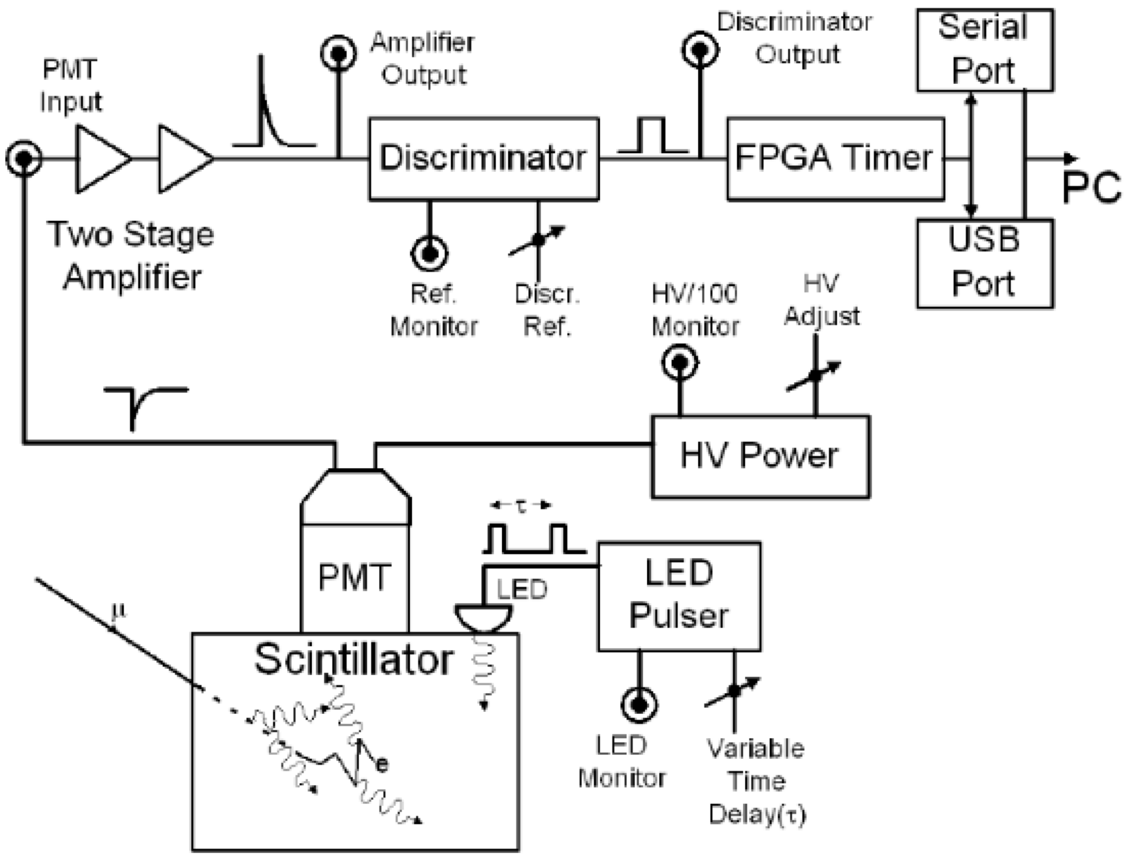
The objective of this portion of the lab was to measure the lifespan of a muon on earth. These results would then be used to calculate the Fermi constant as well as the ratio of positive to negative muons.

Equipment

- Oscilliscope
- Muon electronics box
- Scintillator
- Desktop with Muon software
- Digital Multimeter

Procedure

- Configured the discriminator of the muon electronics box to 210mV
- Adjusted the HV input of the scintillator to 700V
- Connected the Scintillator, Muon box, and computer.



> Figure 9: Configuration of Scintillator, Muon electronics box, and Desktop PC

- Began data collection on the computer
- Waited 30 minutes, and verrified data collected matched the expected Muon spectrum.
- Waited ~3 days for data to be collected, returned and uploaded data onto personal computers.

Data

Data was collected over a slightly less then 3 day period.

In [43]:
```python
data = np.ravel(pd.DataFrame(pd.read_csv("AlexPaul/Section 5/SiftedMuonData.data",sep='

def histoGrammer(data):
    def exp(params,x):
        return params[1] * np.exp(-x/params[0])
    #Making Histogram for 2us
    bins = np.arange(0,20000,2000)
    y,binEdges = np.histogram(data,bins)
    bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    width      = 1900

    #Plotting
    plt.bar(bincenters, y, width=width, color='purple', label = "2 µs bins")

    #Making Histogramfor 1us
    bins = np.arange(0,20000,1000)
    y,binEdges = np.histogram(data,bins)
    bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    width      = 900

    #Plotting
    plt.bar(bincenters, y, width=width, color='red', label = "1 µs bins")

    #Making Histogram for 0.5us
    bins = np.arange(0,20000,500)
    y,binEdges = np.histogram(data,bins)
    bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    width      = 400
    plt.bar(bincenters, y, width=width, color='orange', label = "0.5 µs bins")

    plt.legend()
    plt.title("Histograms of Muon Decay Time with Background Radiation")

    plt.xlabel("Time in ns")
    plt.ylabel("Counts")
    plt.show()

    return

histoGrammer(data)
```

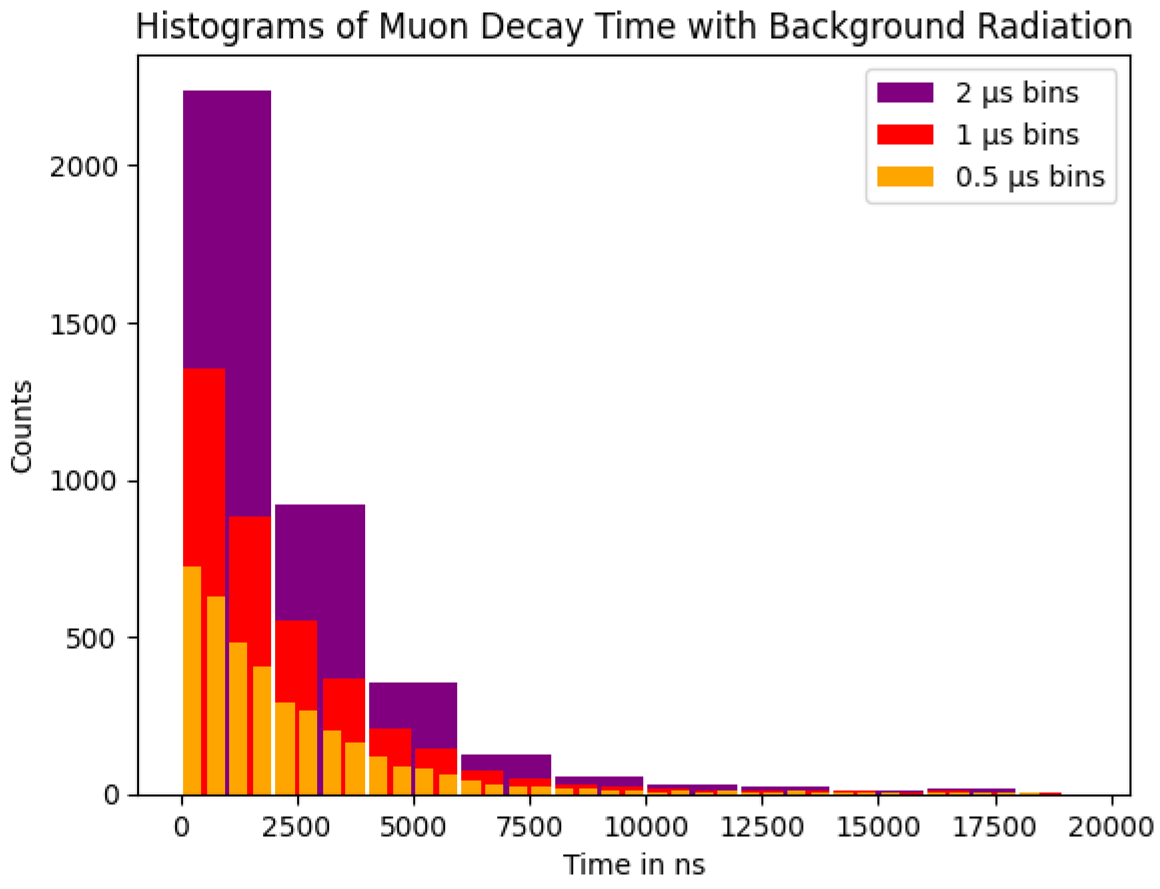
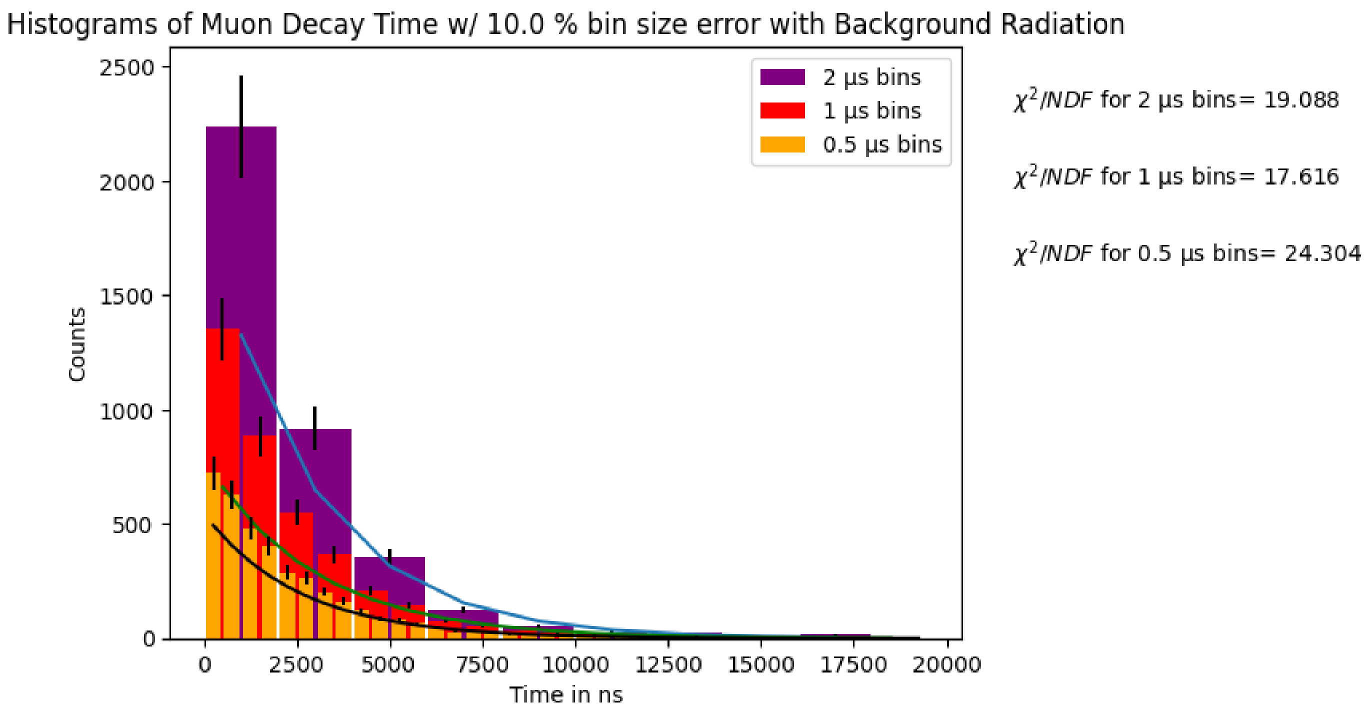


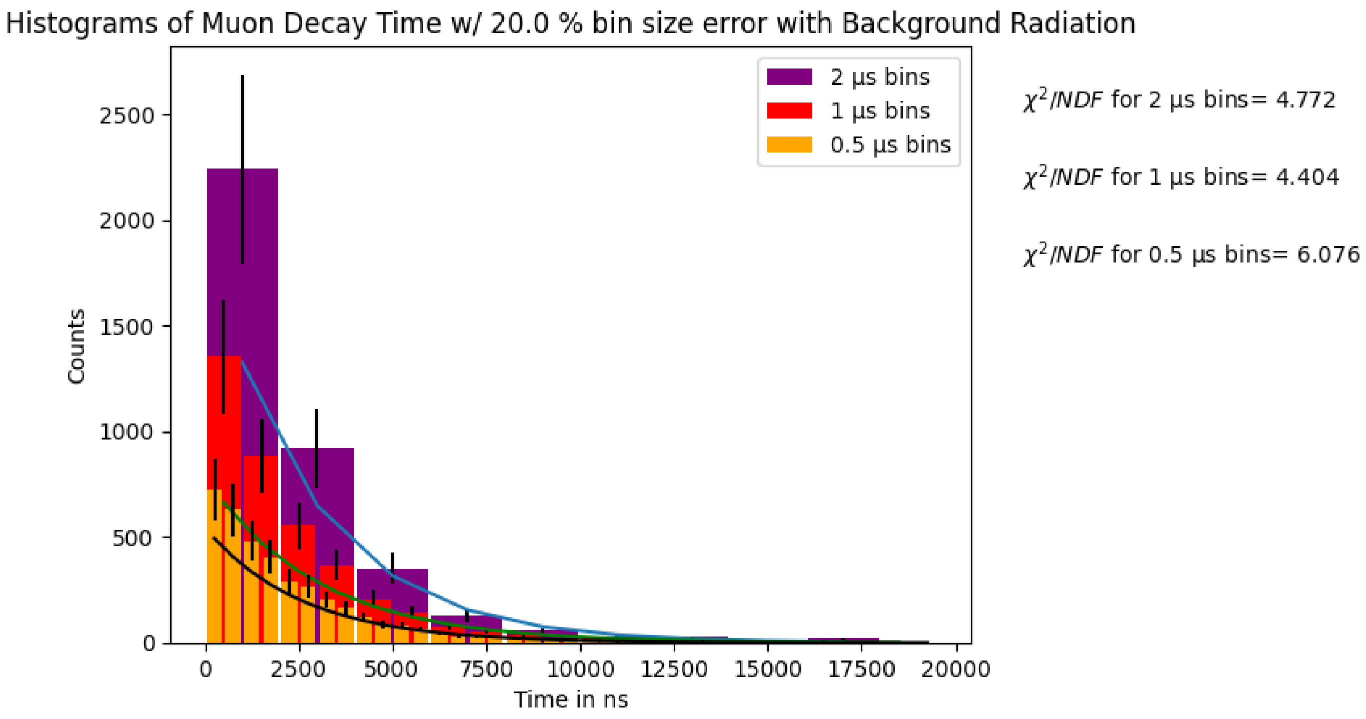
Figure 9.1: Raw Muon lifetime data sorted into 2, 1 and 0.5 us bins

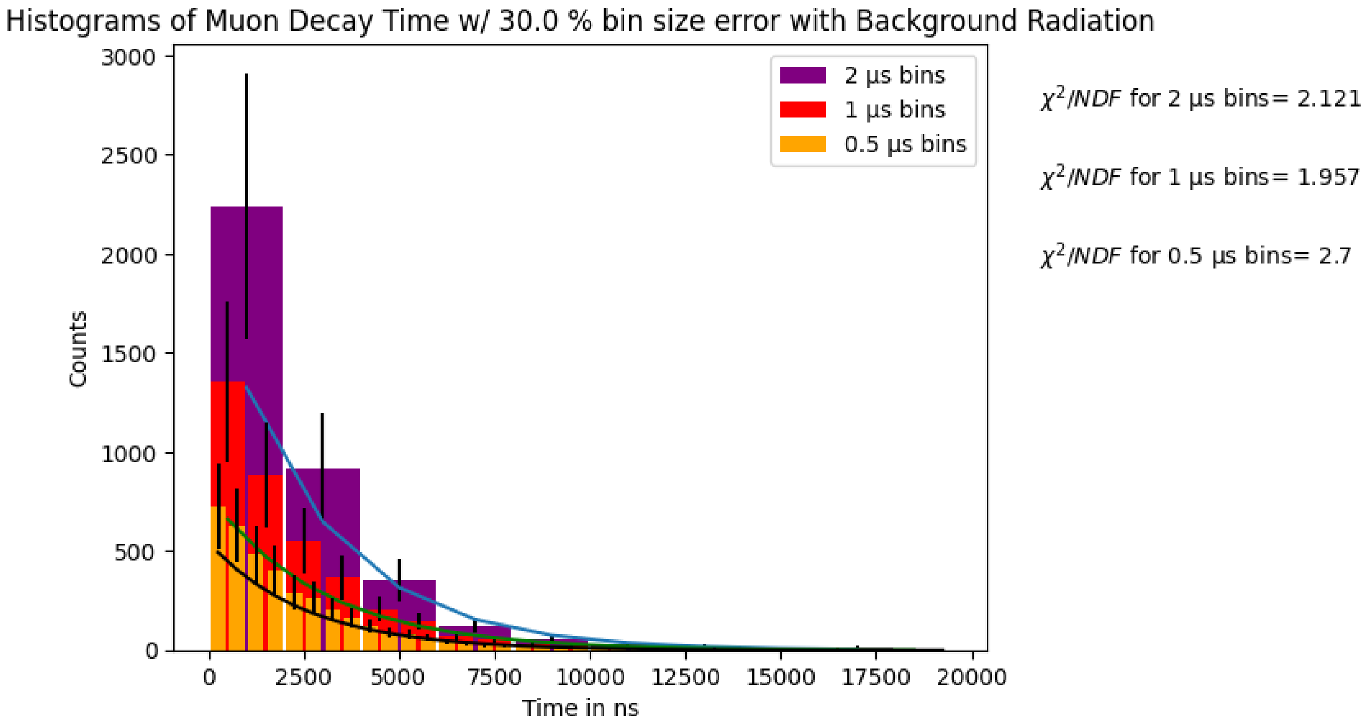
Analysis

Using Method 1: Error of 10,20 and 30 percent of bin size.

In [44]:
```
def histoGrammer(data,errorPrecent, inx1 = 10, inx2 = 20, inx3 = 40):
    def exp(params,x):
        return params[1] * np.exp(-x/params[0])
    model = Model(exp)
    #Making Histogram for 2us
    bins = np.arange(0,20000,2000)
    y,binEdges = np.histogram(data,bins)
    bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    menStd     = y * errorPrecent
    width      = 1900

    #ODR Fit setup for 2us
    odrData        = RealData(bincenters[0:inx1], y[0:inx1],sy = menStd)
    params         = 100,5000
    odr            = ODR(odrData, model, beta0=params)
    odr_result     = odr.run()
    #Extracting Data
    params_fit2 = odr_result.beta
    covar_fit2  = odr_result.sd_beta
    chisquared2 = odr_result.res_var
    V_fit = exp(params_fit2,bincenters)

    #Plotting
    plt.bar(bincenters, y, width=width, color='purple', yerr=menStd, label = "2 µs bins"
    plt.plot(bincenters, V_fit)

    #Making Histogramfor 1us
    bins = np.arange(0,20000,1000)
    y,binEdges = np.histogram(data,bins)
    bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    menStd     = y * errorPrecent
    width      = 900
    #ODR Fit setup for 1us
    odrData        = RealData(bincenters[0:inx2], y[0:inx2],sy = menStd)
    params         = 100,2000
    odr            = ODR(odrData, model, beta0=params)
    odr_result     = odr.run()

    #Extracting Data
    params_fit1 = odr_result.beta
    covar_fit1  = odr_result.sd_beta
    chisquared1 = odr_result.res_var
    V_fit = exp(params_fit1,bincenters)

    #Plotting
    plt.bar(bincenters, y, width=width, color='red', yerr=menStd, label = "1 µs bins")
    plt.plot(bincenters, V_fit, color = "green")

    #Making Histogram for 0.5us
    bins = np.arange(0,20000,500)
    y,binEdges = np.histogram(data,bins)
    bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    menStd     = y * errorPrecent
    width      = 400
    #ODR Fit setup for 0.5us
    odrData        = RealData(bincenters[0:inx3], y[0:inx3],sy = menStd)
    params         = 100,2000
    odr            = ODR(odrData, model, beta0=params)
    odr_result    = odr.run()

    #Extracting Data
    params_fit05 = odr_result.beta
    covar_fit05  = odr_result.sd_beta
    chisquared05 = odr_result.res_var
    V_fit = exp(params_fit05,bincenters)
    plt.bar(bincenters, y, width=width, color='orange', yerr=menStd, label = "0.5 µs bin
    plt.plot(bincenters, V_fit, color = "black")

    plt.legend()
    plt.title("Histograms of Muon Decay Time w/ " + str(errorPrecent * 100) + " % bin si
    plt.gcf().text(0.95,0.8, "$\\chi^2/NDF$ for 2 µs bins= " + str(np.round(chisquared2,
    plt.gcf().text(0.95,0.7, "$\\chi^2/NDF$ for 1 µs bins= " + str(np.round(chisquared1,
    plt.gcf().text(0.95,0.6, "$\\chi^2/NDF$ for 0.5 µs bins= " + str(np.round(chisquared
    plt.xlabel("Time in ns")
    plt.ylabel("Counts")
    plt.show()

    paramsArray = np.array([params_fit05,params_fit1,params_fit2])
    chisquaredArray = chisquared05, chisquared1, chisquared2
    return paramsArray, chisquaredArray

# Saving Params to array
p1,c1 = histoGrammer(data,0.1)
p2,c2 = histoGrammer(data,0.2)
p3,c3 = histoGrammer(data,0.3)
```





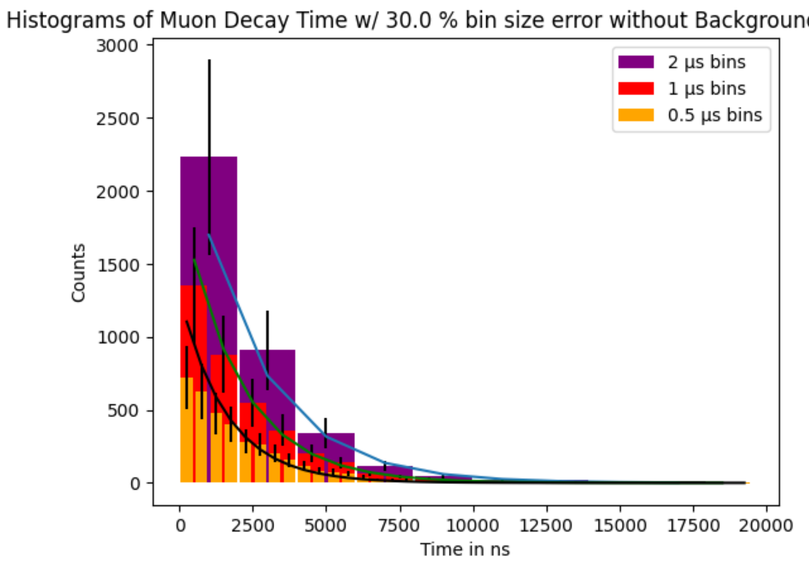


> Figures 10, 11, 12: Histograms of muon data with 10, 20 and 30 percent error for each bin repsectively

Removing Background Radiation

In [45]:

```python
#Returns counts that have a value greater then tennative x 5
def trimmer(tennative, bins, counts):
    # Calculate the threshold
    ten5 = tennative * 5

    # Find the index where bins first exceed ten5
    index = np.argmax(bins > ten5)

    # Check if the index is valid
    if index == 0 and not np.any(bins > ten5):
        raise ValueError("No bins exceed the threshold value.")
    # Slice the bins and counts arrays
    longBins = bins[index:]
    longCounts = counts[index:]

    # Check the lengths of the resulting arrays
    if len(longBins) != len(longCounts):
        raise ValueError("Sliced bins and counts arrays are not the same length.")

    return longBins, longCounts

#Returns a lifetime from a set of data with error
def lifetime(bins,counts,error):
    def exp(params,x):
        return params[1] * np.exp(-x/params[0])
    errorPrecent = error
    model = Model(exp)
    menStd     = counts * errorPrecent
    #ODR Fit setup
    odrData        = RealData(bins, counts,sy = menStd)
    params         = 100,2000
    odr            = ODR(odrData, model, beta0=params)
    odr_result     = odr.run()
    #Extracting Data
    params = odr_result.beta
    covar = odr_result.sd_beta
    return params[0], covar[0]

#Returns a scalar background level
def linearCutoff(bins,counts):
    def line(x, params):
        return params
    pc1,pt = curve_fit(line, bins, counts)
    background = np.average([pc1])
    return background

#Returns histogrammable data with background removed through 3 step process
def backgroundRemove(step,data,error):
    bins = np.arange(0,20000,step)
    counts ,binEdges = np.histogram(data,bins)
    bins = 0.5*(binEdges[1:]+binEdges[:-1])
    t1 = lifetime(bins,counts,error)[0]
    lb1, lc1 = trimmer(t1,bins, counts)
    backgroundLevel1 = linearCutoff(lb1, lc1)
    counts1 = np.subtract(counts, backgroundLevel1)
    t2 = lifetime(bins,counts1,error)[0]
    lb2, lc2 = trimmer(t2, bins, counts1)
    backgroundLevel2 = linearCutoff(lb2, lc2)
    counts2 = np.subtract(counts, backgroundLevel2)
    t3 = lifetime(bins,counts2,error)[0]
    lb3, lc3 = trimmer(t3,bins, counts2)
    backgroundLevel3 = linearCutoff(lb3, lc3)
    counts3 = np.subtract(counts, backgroundLevel3)
    t4 = lifetime(bins,counts3,error)
    return(bins, counts3,t4)



def histoGrammerMod(data,errorPrecent):
    def exp(params,x):
        return params[1] * np.exp(-x/params[0])
    model = Model(exp)
    #Making Histogram for 2us
    bins,counts,time2 = backgroundRemove(2000, data,errorPrecent)
    menStd     = np.abs(counts * errorPrecent)
    width      = 1900
    #ODR Fit setup for 2us
    odrData        = RealData(bins, counts ,sy = menStd)
    params         = 100,3000
    odr            = ODR(odrData, model, beta0=params)
    odr_result     = odr.run()
    #Extracting Data
    params_fit2 = odr_result.beta
    covar_fit2  = odr_result.sd_beta
    chisquared2 = odr_result.res_var
    V_fit = exp(params_fit2,bins)

    #Plotting
    plt.bar(bins, counts, width=width, color='purple', yerr=menStd, label = "2 µs bins")
    plt.plot(bins, V_fit)

    #Making Histogramfor 1us
    bins,counts,time1 = backgroundRemove(1000, data,errorPrecent)
    menStd      = np.abs(counts * errorPrecent)
    width       = 900
    #ODR Fit setup for 1us
    odrData        = RealData(bins, counts ,sy = menStd)
    params         = 100,1400
    odr            = ODR(odrData, model, beta0=params)
    odr_result     = odr.run()

    #Extracting Data
    params_fit1 = odr_result.beta
    covar_fit1  = odr_result.sd_beta
    chisquared1 = odr_result.res_var
    V_fit = exp(params_fit1,bins)

    #Plotting
    plt.bar(bins, counts, width=width, color='red', yerr=menStd, label = "1 µs bins")
    plt.plot(bins, V_fit, color = "green")

    #Making Histogram for 0.5us
    bins,counts,time05 = backgroundRemove(500, data,errorPrecent)
    menStd      = np.abs(counts * errorPrecent)
    width       = 400
    #ODR Fit setup for 0.5us
    odrData        = RealData(bins, counts ,sy = menStd)
    params         = 100,600
    odr            = ODR(odrData, model, beta0=params)
    odr_result     = odr.run()

    #Extracting Data
    params_fit05 = odr_result.beta
    covar_fit05  = odr_result.sd_beta
    chisquared05 = odr_result.res_var
    V_fit = exp(params_fit05,bins)
    plt.bar(bins, counts, width=width, color='orange', yerr=menStd, label = "0.5 µs bins
    plt.plot(bins, V_fit, color = "black")

    plt.legend()
    plt.title("Histograms of Muon Decay Time w/ " + str(errorPrecent * 100) + " % bin si
    plt.gcf().text(0.95,0.8, "$\\chi^2/NDF$ for 2 µs bins= " + str(np.round(chisquared2,
    plt.gcf().text(0.95,0.7, "$\\chi^2/NDF$ for 1 µs bins= " + str(np.round(chisquared1,
    plt.gcf().text(0.95,0.6, "$\\chi^2/NDF$ for 0.5 µs bins= " + str(np.round(chisquared
    plt.gcf().text(0.95,0.5, "Calculated Decay time for 2 µs bins $\\pm$= " + str(np.rou
    plt.gcf().text(0.95,0.4, "Calculated Decay time for 1 µs bins $\\pm$ = " + str(np.ro
    plt.gcf().text(0.95,0.3, "Calculated Decay time for 0.5 µs bins $\\pm$= " + str(np.r
    plt.xlabel("Time in ns")
    plt.ylabel("Counts")
    plt.show()

    paramsArray = np.array([params_fit05,params_fit1,params_fit2]), np.array([covar_fit0
    chisquaredArray = chisquared05, chisquared1, chisquared2
    return paramsArray, chisquaredArray

params3, chi3 = histoGrammerMod(data, 0.3)
params2, chi2 = histoGrammerMod(data, 0.2)
params1, chi1 = histoGrammerMod(data, 0.1)
```



χ^2/NDF for 2 µs bins= 1.614

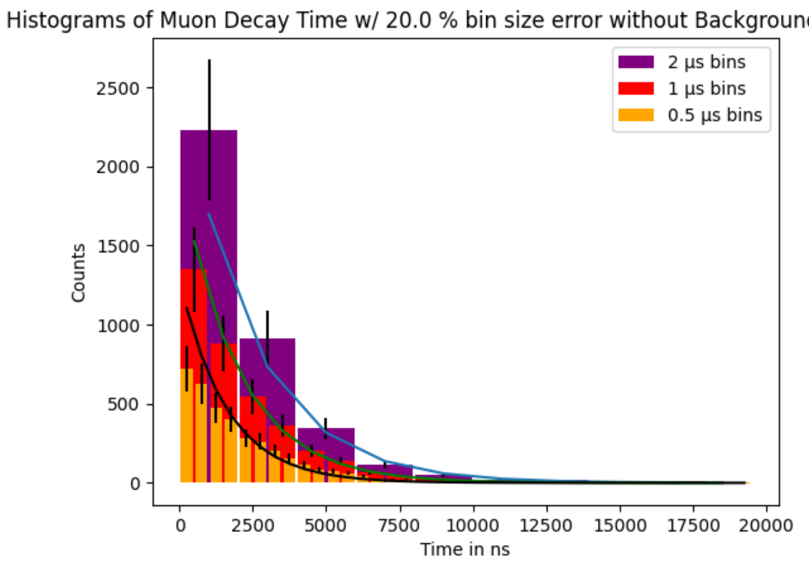
χ^2/NDF for 1 µs bins= 3.906

χ^2/NDF for 0.5 µs bins= 6.838

Calculated Decay time for 2 µs bins ±= [2382.887 196.217]

Calculated Decay time for 1 µs bins ± = [1962.826 149.962]

Calculated Decay time for 0.5 µs bins ±= [1586.865 134.652]



χ^2/NDF for 2 µs bins= 3.63

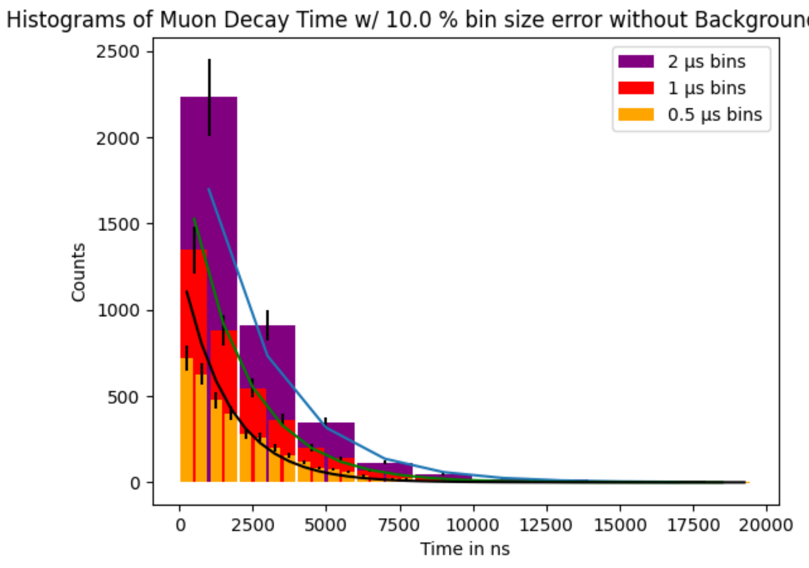
χ^2/NDF for 1 µs bins= 8.788

χ^2/NDF for 0.5 µs bins= 15.384

Calculated Decay time for 2 µs bins ±= [2382.887 196.217]

Calculated Decay time for 1 µs bins ± = [1962.826 149.962]

Calculated Decay time for 0.5 µs bins ±= [1586.866 134.652]



χ^2/NDF for 2 µs bins= 14.521

χ^2/NDF for 1 µs bins= 35.152

χ^2/NDF for 0.5 µs bins= 61.537

Calculated Decay time for 2 µs bins ±= [2382.888 196.216]

Calculated Decay time for 1 µs bins ± = [1962.829 149.964]

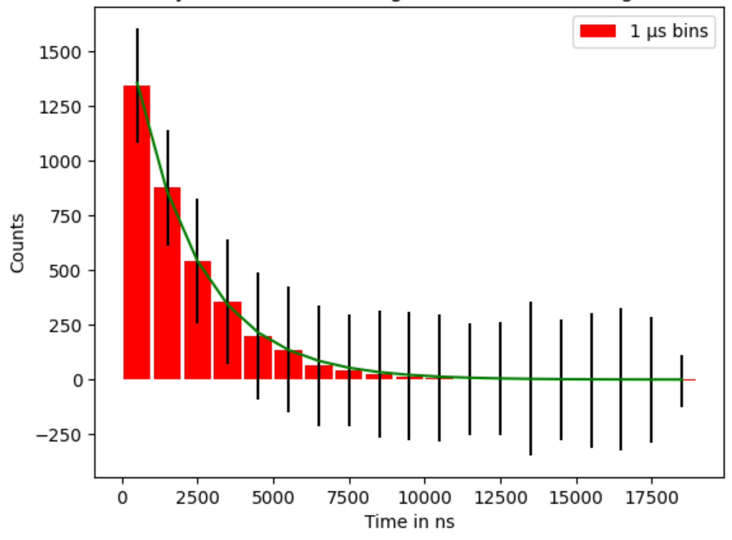
Calculated Decay time for 0.5 µs bins ±= [1586.867 134.654]

> Figures 13,14,15: Histograms of muon data with background radiation removed, at error levels 30,20,10 respectively

Method 2: Using bin size to calcuilate Standard Deviation and using this as error size. Picking 1us bin size:

In [46]:
```
def histogram_with_data(data, bins):
    counts, bin_edges = np.histogram(data, bins=bins)
    binned_data = []
    for i in range(len(bin_edges) - 1):
        bin_mask = (data >= bin_edges[i]) & (data < bin_edges[i + 1])
        binned_data.append(data[bin_mask])
        bin_mask = data == bin_edges[-1]
    binned_data[-1] = np.concatenate((binned_data[-1], data[bin_mask]))
    return binned_data, counts, bin_edges

def lifetimeMod(bins,counts,error):
    def exp(params,x):
        return params[1] * np.exp(-x/params[0])
    model = Model(exp)
    #ODR Fit setup
    odrData       = RealData(bins, counts,sy = error)
    params        = 100,2000
    odr           = ODR(odrData, model, beta0=params)
    odr_result    = odr.run()
    #Extracting Data
    params = odr_result.beta
    covar = odr_result.sd_beta
    return params[0], covar[0]

def backgroundRemoveMod(step,data,errorArr):
    bins = np.arange(0,20000,step)
    counts ,binEdges = np.histogram(data,bins)
    bins = 0.5*(binEdges[1:]+binEdges[:-1])
    t1 = lifetimeMod(bins,counts,errorArr)[0]
    lb1, lc1 = trimmer(t1,bins, counts)
    backgroundLevel1 = linearCutoff(lb1, lc1)
    counts1 = np.subtract(counts, backgroundLevel1)
    t2 = lifetimeMod(bins,counts1,errorArr)[0]
    lb2, lc2 = trimmer(t2, bins, counts1)
    backgroundLevel2 = linearCutoff(lb2, lc2)
    counts2 = np.subtract(counts, backgroundLevel2)
    t3 = lifetimeMod(bins,counts2,errorArr)[0]
    lb3, lc3 = trimmer(t3,bins, counts2)
    backgroundLevel3 = linearCutoff(lb3, lc3)
    counts3 = np.subtract(counts, backgroundLevel3)
    t4 = lifetimeMod(bins,counts3,errorArr)
    return(bins, counts3,t4)

def histoGrammerModMod(data,errArr):
    def exp(params,x):
        return params[1] * np.exp(-x/params[0])
    model = Model(exp)
    #Making Histogramfor 1us
    bins,counts,time1 = backgroundRemoveMod(1000, data,errArr)
    width       = 900
    #ODR Fit setup for 1us
    odrData       = RealData(bins, counts ,sy = errArr)
    params        = 100,1400
    odr           = ODR(odrData, model, beta0=params)
    odr_result    = odr.run()

    #Extracting Data
    params_fit1 = odr_result.beta
    covar_fit1  = odr_result.sd_beta
    chisquared1 = odr_result.res_var
    V_fit = exp(params_fit1,bins)

    #Plotting
    plt.bar(bins, counts, width=width, color='red', yerr=errArr, label = "1 µs bins")
    plt.plot(bins, V_fit, color = "green")

    plt.legend()
    plt.title("Histograms of Muon Decay Time without Background Radiation using Standard
    plt.gcf().text(0.95,0.7, "$\\chi^2/NDF$ for 1 µs bins= " + str(np.round(chisquared1,
    plt.gcf().text(0.95,0.6, "Calculated Decay time for 1 µs bins = " + str(np.round(tim
    plt.xlabel("Time in ns")
    plt.ylabel("Counts")
    plt.show()
    print(time1)
    params = params_fit1,covar_fit1
    return params, chisquared1


bins = np.arange(0,20000,1000)
dataBins, counts, bin_edges = histogram_with_data(data,bins)
stdDevs = []
for i in range(len(dataBins)):
    stdDevs.append(np.std(dataBins[i-1]))

params4, chi4  = histoGrammerModMod(data, stdDevs)
```

Histograms of Muon Decay Time without Background Radiation using Standard Devation of Bins



χ^2/NDF for 1 µs bins= 0.002

Calculated Decay time for 1 µs bins = 2173.951±29.538

```
(2173.9509939065156, 29.538065682663856)
```

> Figure 16: Muon histogram using 1us bins and standard deviation as bin error

Making table for all error values and bin sizes for Method 1

In [47]:
```
times1 = params1[0][:, 0]
times2 = params2[0][:, 0]
times3 = params3[0][:, 0]
unc1 = params1[1][:, 0]
unc2 = params2[1][:, 0]
unc3 = params3[1][:, 0]
chiArr = np.array([chi1,chi2,chi3])
timesArr = np.array([times1, times2, times3])
uncArr = np.array([unc1, unc2, unc3])
row_labels = ['10% error', '20% error', '30% error']
col_labels = ['0.5 us bins', '1 us bins', '2 us bins']
formatted_matrix = np.empty(timesArr.shape, dtype=object)
for i in range(timesArr.shape[0]):
    for j in range(timesArr.shape[1]):
        formatted_matrix[i, j] = f"{timesArr[i, j]:.4f} ± {uncArr[i, j]:.4f}"
table = np.empty((4, 4), dtype=object)
table[0, 1:] = col_labels
```

```
table[1:, 0] = row_labels
table[1:, 1:] = formatted_matrix
table[0, 0] = " "
mpr.params.title = False
mpr.params.indices = False

table
```

Out[47]:

	0.5 us bins	1 us bins	2 us bins
10% error	1586.8169 ± 134.6546	1962.8274 ± 149.9635	2382.9180 ± 196.2218
20% error	1586.8158 ± 134.6532	1962.8248 ± 149.9623	2382.9166 ± 196.2220
30% error	1586.8156 ± 134.6530	1962.8243 ± 149.9621	2382.9163 ± 196.2220

Figure 17: Table of muon lifetimes for 0.5, 1 and 2us bins, at error level 10,20 and 30%

In [14]:
```
chiTable = np.empty((4, 4), dtype=object)
chiTable[0, 1:] = col_labels
chiTable[1:, 0] = row_labels
for i in range(chiArr.shape[0]):
    for j in range(chiArr.shape[1]):
        chiTable[i + 1, j + 1] = f"{chiArr[i, j]:.2f}"
chiTable[0, 0] = " "

chiTable
```

Out[14]:

	0.5 us bins	1 us bins	2 us bins
10% error	61.54	35.15	14.52
20% error	15.38	8.79	3.63
30% error	6.84	3.91	1.61

Figure 18: Table of chi squared values

Analysis

Next, we are to investigate the Fermi Coupling constant using the muon decay time we derived using method 2. The relation is given below

$$\tau = \frac{192\pi^3\hbar^7}{G_F^2 m^5 c^4}$$

Where τ is the lifetime of the muon and G_F is the Fermi Coupling constant. Rearranging this to solve for the Fermi constant we get the following:

$$G_F^2 = \frac{192\pi^3\hbar^7}{\tau m^5 c^4}$$

Converting this into the desired final form:

$$\frac{G_F}{(\hbar c)^3} = \sqrt{\frac{1}{\tau}}\sqrt{\frac{\hbar 192\pi^3}{(mc^2)^5}}$$

Mass of muon: 0.105 $\frac{GeV}{c^2}$ $\hbar = 6.582E - 25$

Solving constant term:

$$\sqrt{\frac{\hbar 192\pi^3}{(m_\mu c^2)^5}} = 1.75219 \times 10^{-8}$$

Now we can solve this with uncertainty:

$$\frac{G_F}{(\hbar c)^3} = 0.0000118838 \qquad \begin{aligned} \Delta G_F &= \sqrt{\left(\frac{\partial G_F}{\partial t}\Delta t\right)^2} \\ &= \sqrt{\left((-(1.75219e-8/(2*sqrt(1/t))/t^2)) \cdot \Delta t\right)^2} \\ &= 8.073e-7 \\ &= 8.073 \times 10^{-7} \\ &= 8 \times 10^{-7} \end{aligned}$$

$$\frac{G_F}{(\hbar c)^3} = (1.19 \pm 0.08) \times 10^{-5}$$

*credit to https://pdg.lbl.gov/2020/listings/rpp2020-list-muon.pdf particle data group for muon mass

Next, we are to solve for ρ, the ratio of positive to negative muons. This ratio is given by:

$$\rho = -\frac{\tau^+}{\tau^-}\left(\frac{\tau^- - \tau_{\text{obs}}}{\tau^+ - \tau_{\text{obs}}}\right)$$

Where τ^+ is the decay rate of positively charged muons, identical to the free-space decay rate $\tau = 2.197 \pm 0.00004\mu s$. τ^- is the decay time of negatively charged muons, which decay faster in matter. The decay rate of these particles in matter is given by $\tau_c = 2.043 \pm 0.003\mu s$

Using our observed muon decay time of

$$\tau_{\text{obs}} = 2.173 \pm 0.029$$

We can solve for rho:

$$\rho$$

$$= 5.8249714472 \qquad \begin{aligned} \Delta\rho &= \sqrt{\left(\frac{\partial\rho}{\partial t}\Delta t\right)^2 + \left(\frac{\partial\rho}{\partial tp}\Delta tp\right)^2 + \left(\frac{\partial\rho}{\partial tm}\Delta tm\right)^2} \\ &= \sqrt{\begin{array}{l}\left((-(tp*1/tm/(tp-t)^2*(tm-tp))) \cdot \Delta t\right)^2 \\ + \left((tp*1/tm/(tp-t)^2*(tm-t)-(tm-t)*1/tm/(tp-t)^1) \cdot \Delta tp\right)^2 \\ + \left((tp*1/(tp-t)/tm^2*(tm-t)-tp*1/(tp-t)/tm^1) \cdot \Delta tm\right)^2\end{array}} \\ &= 8.3391551635 \\ &= 8.3391551635 \times 10^0 \\ &= 8 \times 10^0 \end{aligned}$$

$$\rho = 6 \pm 8$$

Conclusion:

Through this section we calculated the lifetime of a muon, the Fermi Constant, and the ratio of positive to negative muons that landed within our collector. Our values for muon lifetime were slightly lower then expected but this can also be attributed to negative muons decaying in matter quicker. If we were able to detect muons in free space, we would be able to derive the free space Muon decay time. Our best fitting muon decay time using method 1 was 2382.9163 ± 196.2220 nanoseconds, with a reduced chi squared of 1.61. This is within uncertainty of the literature value of muon lifetime, however, knowing that the detected muon lifetime should be slightly less, this is slightly off. An explanation for this is the approach utilized a flat rate error, which heavily weighted bins with fewer data-points. In an exponentially decaying model like this one, this means that the higher-value bins were much more heavily weighted, increasing the overall calculated muon lifetime. Using a much better approach, taking the standard deviation of each bin, we derived a muon decay time of 2173.951 $\pm$ 29.538 nanoseconds, slightly below the free space lifetime as predicted. However, despite this, we had a highly over-constrained graph due to the large errors in the latter part of the spectrum. This is evidenced by the derived reduced chi squared of 0.002. Using this second value, we calculated the Fermi constant and muon ratio ρ. These values were respectively calculated at $\frac{G_F}{(\hbar c)^3} = (1.19 \pm 0.08) \times 10^{-5}$ and $\rho = 6 \pm 8$, both falling within literature estimated values. The Muon ratio had a large error due to all three muon decay times, both given and derived, having non-negligible errors, resulting in a relatively large propagated error.