physics@mit

MEMORANDUM

To: Bobby Johnston

From: Peter Fisher

Subject: Interface for NaI digitizer readout

Date: January 23, 2018

I have a utility called `wavedump` that readout out the CAEN 6720 digitizer, does the pulseheight analysis for each pulse, and fills a 4096 channel histogram with the results. The histogram is written out every second to `/tmp/PlotData.txt.` `wavedump` is written by CAEN and is robust, so we do not want to make many changes to it. This memo describes the display and recording program requirements. This program should be written in root and run on the same machine as `wavedump`.

1 PMT and analog signal chain

The NaI crystal is a right cylinder of 2 inches diameter and 2 inches thickness. The crystal is mounted on a PMT/Amplifier/HV supply package (Ortec 905-3). The PMT base has two outputs: an voltage output from the amplifier with a 4 μs shaping time and a signal from the last dynode. There are two controls: an on/off push button switch and a potentiometer that set the high voltage. The HV setting may be measured from a test point with a DVM. The measured voltage in mV is the set voltage in V. The last dynode may be thought of as a capacitor with a capacitance large compared with the input capacitance of the CAEN 6720. The charge arrives at the last dynode in a 40 ns time interval and we want to sample a small fraction of that charge with the digitizer. Fig. **??** shows the equivalent circuit. We think of a current pulse arriving over 40 ns, charging the two capacitors with a time constant of $1/R\,(C+C')$. Then the pulse ends, the switch opens, and the input capacitance C discharges with time constant $1/RC$. The digitizer records the voltage V_{in} every 4 ns.

The pulse shape recorded by the digitizer may be modeled as,

$$i\,(t) = \begin{array}{cl} 0 & t < t_1 \\ A\left(1 - e^{-\lambda_1 (t-t_1)}\right) & t_1 < t < t_2 \\ A\left(1 - e^{-\lambda_1 (t_2-t_1)}\right) e^{-\lambda_2 (t-t_2)} & t_2 < t. \end{array} \qquad (1)$$

The integral of the charge collected is given by,

$$Q \;=\; A\,(t_2 - t_1) + \left(\frac{A}{\lambda_2} - \frac{A}{\lambda_1}\right)\left(1 - e^{-\lambda (t_2 - t_1)}\right). \qquad (2)$$

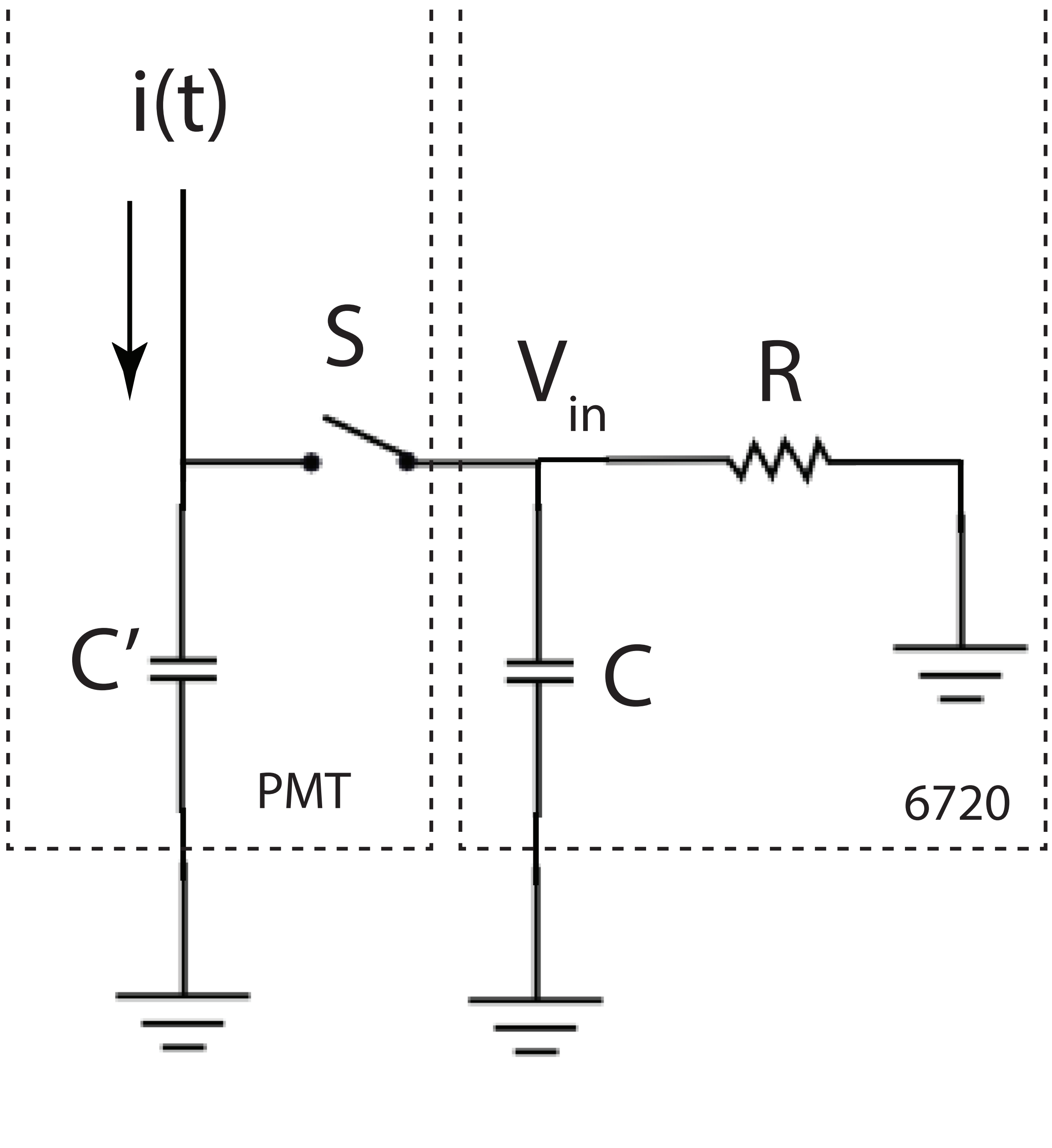


Figure 1: Equivalent circuit for last dynode of PMT and input of CAEN 6720.

2 CAEN 6720 waveform digitizer operation and `wavedump` output

`wavedump` setups up the 6720 digitizer and, upon user command, begins data collection. `wavedump` reads 1024 samples at 250 MHz, corresponding to a 4.1 μs sample window, and processes the results. `wavedump` setup is determined by a configuration file
`/home/fisherp/projects/CAEN/wavedump-3.8.1/Setup/myWaveDump`
that set the trigger threshold, readout frequency, data range, etc. In the current configuration, the input signal from the pre-amp of the NaI crystal is -1 to 1 V into 50 Ω. This means each sample is the current from the last dynode into 50 Ω integrated over 4 ns. The 6720 has 12 bit ADC, so one channel is 48 μV or 38.4 fC. The total charge is the sum of all the readout channels.

The baseline of the 6720 is set at approximately channel 2048. Currently, `wavedump` uses the difference between to largest and smallest of the 1024 readout bins as an estimator for the total charge. This will be improved later. For each pulse `wavedump` estimates the total charge and bins the result into a 4096 channel histogram. The histogram is written out to `/tmp/Plotdata.txt` every second. The histogram accumulates until `wavedump` exits. `PlotData.txt` is an ASCII file with two columns of numbers: the first is channel number and the second is accumulated contents.

3 `wavedump` operation

4 Extraction of the charge integral from the recorded waveform

Fig. 2 shows a typical event collected from the NaI detector by the digitizer. The energy deposited in the NaI crystal is proportional to the total charge flowing from the PMT, Q. We can estimate Q from the waveform in three different ways.

1. Peak height - this has the advantage of being very simple and fast. However, resolution suffers as only a single point is used for the estimate and there is no quality control on the shape of the pulse.

2. Integral under the peak - in this case all the points (about 300) are used in the estimate, as well as 150 points from before the trigger. This methods is nearly as fast as peak finding. There is no quality control on the pulse shape.

3. Peak fit - in this case the entire pulse up to channel 450 use used on a six parameter fit to Eq. 1 (the five parameters in Eq. 1 and the position of the baseline near channel 2048.) This method gives a goodnes of fit parameter, as well as all six parameters of the pulse shape and uses all the information available. Fitting is slow, 40 ms in root interpretive mode.

The different methods give different uncertainties. The variance of the baseline noise is around 2.7 counts^2, Fig. **??**. For a typical peak amplitude of 20 counts, the square root over the variance gives a fractional uncertainty of 6.5%. This is on top of the the uncertainties of in the counting statistics of the scintillation photons that determine the intrinsic resolution of the detector. Using 100 eV/photon, a quantum efficiency of 20%, and a geometric collection fraction of 25%, for 3 MeV energy deposition, we expect 1,500 photons. Using counting statistics alone, this would give 2.5% resolution.

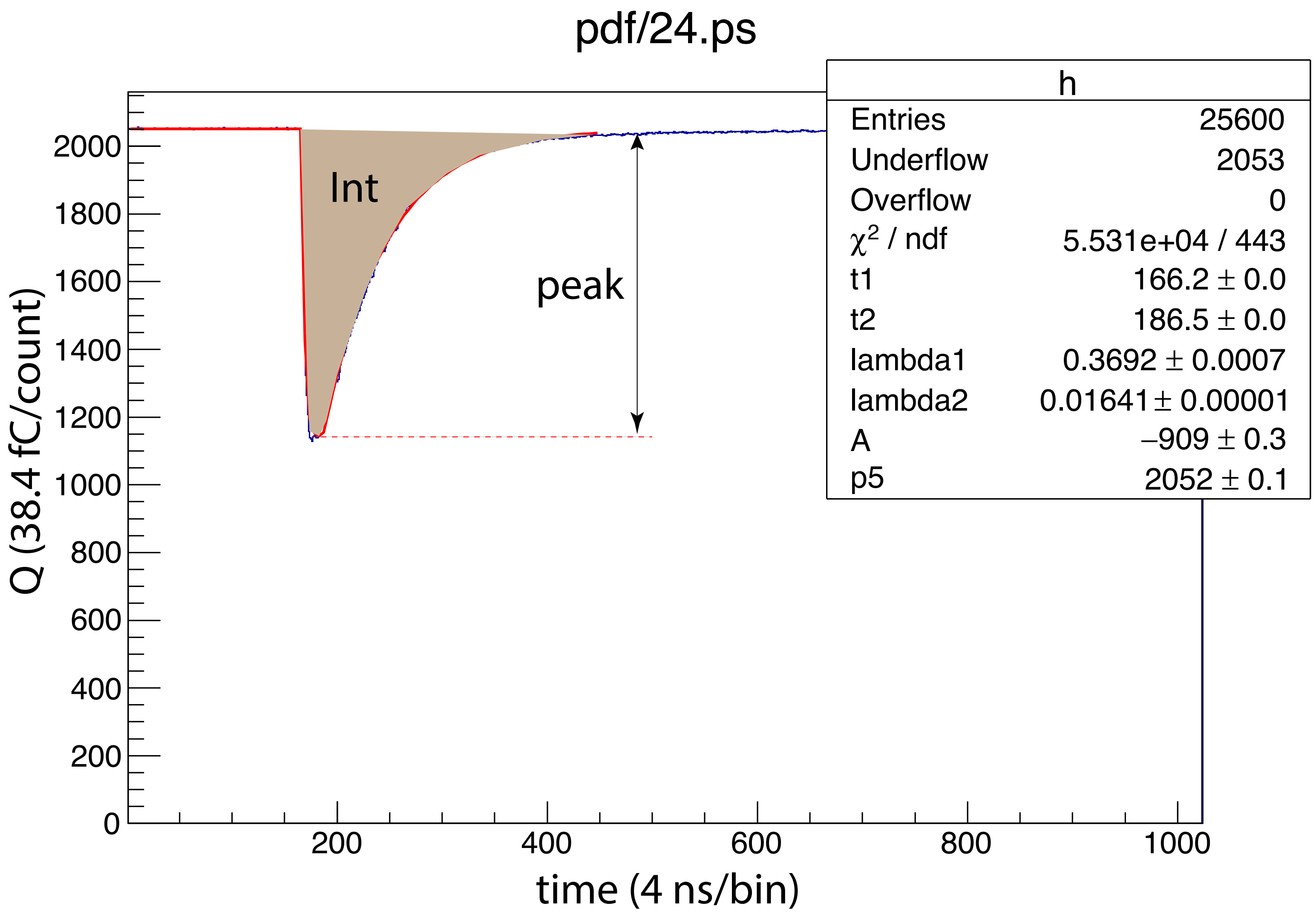


Figure 2: Waveform for a larger than usual pulse recorded by the CAEN 6720. The total integral $Q = -3701 \pm 39$ units (38.9 fC/unit) using the fit. Integration gives $Q = -3669 \pm 30$ units.

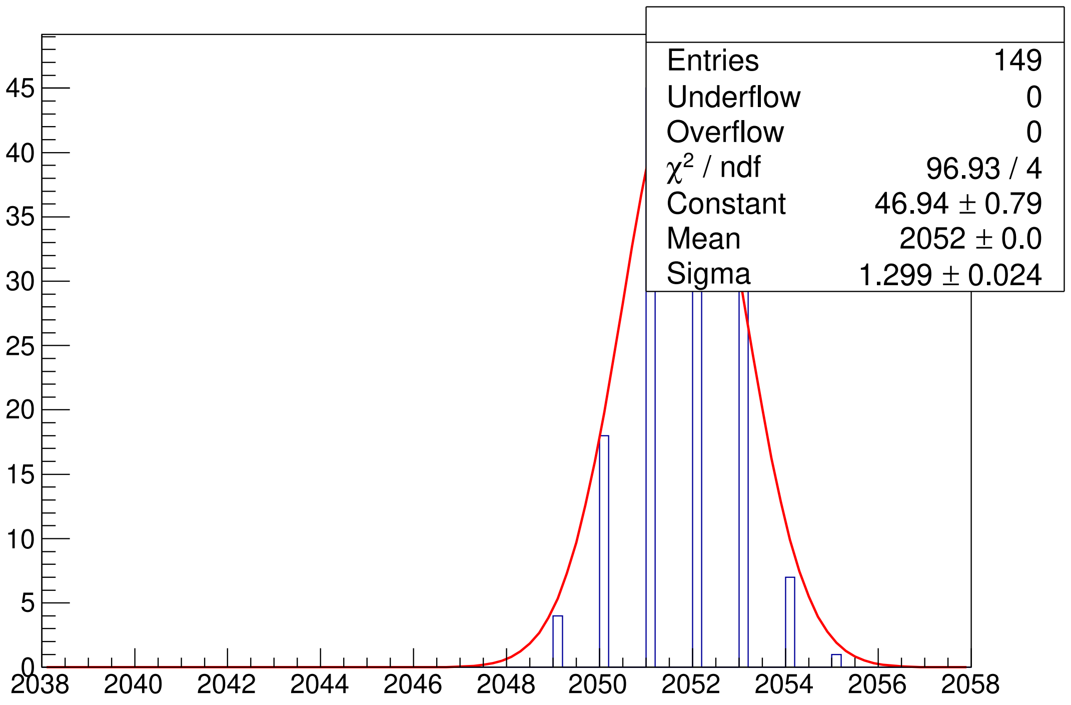


Figure 3: Binning of first 150 time bins of baseline noise from a typical event. Fit is for a Gaussian.

For the integration routine runs over 300 bins, which gives an error on the charge uncertainty of 28 counts for $Q = 9000$ counts, contributing 0.3%. For the fit, we use the expression in Eq. 2, the uncertainties extracted from the fit, and the standard uncertainty relation,

$$\sigma_Q = \sqrt{\left(\frac{\partial Q}{\partial t_1}\right)^2 \sigma_{t_1}^2 + \left(\frac{\partial Q}{\partial t_2}\right)^2 \sigma_{t_2}^2 + \left(\frac{\partial Q}{\partial \lambda_1}\right)^2 \sigma_{\lambda_1}^2 + \left(\frac{\partial Q}{\partial \lambda_2}\right)^2 \sigma_{\lambda_2}^2 + \left(\frac{\partial Q}{\partial A}\right)^2 \sigma_A^2 +},$$

and get about the same uncertainty. However, the fit also gives a χ^2 statistic tells us how good the fit is and this may help remove noisy or overlapping events.

Fig. 4 shows the Q extraction performance for 10 minutes of data. For both the integration and peak methods, the ^{208}Tl 2614 keV line is clearly visible new $Q = 9000$ in the fit. In the expanded plot, the maximum MIP energy lies around 90,000 or 35 MeV. This comes from a NaI density of 2.7 g/cm^3, $dE/dx_{min} = 1.3$ MeV/g/cm^2, and a typical path length of $sqrt2 \times 5$ cm=7 cm. A fit of the 2614 keV line gives a fractional resolution of 2.5%, consistent with expectation, Fig. 5.

5 Display program specifications

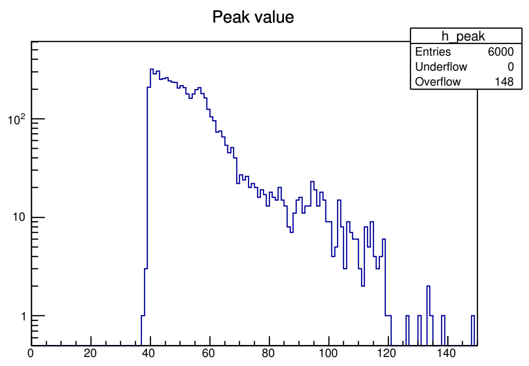
The display program has the following functions:

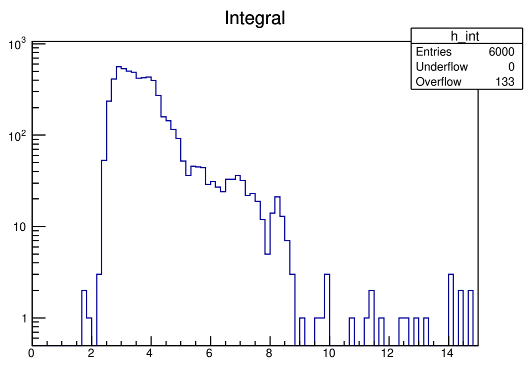
1. Read the histogram every 5s from `/tmp/PlotData.txt`. The exact time the histogram is read out is not important, but it should be timestamped using `time()`.
2. Record each histogram with it time stamp as a `TH1F` object in a root file.
3. Display each histogram as it is acquired in a continuously updating display.
4. Display difference histograms or other accumulated histograms as useful. All displays should be in a single `TCanvas`.
5. Allow operator control of the display in real time using the `kbhit()` command.

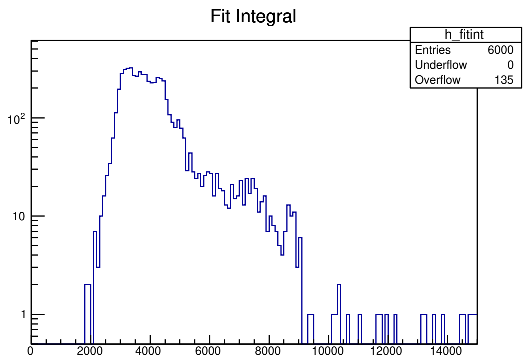
The display program may be a root macro or standalone program. Eventually, it should also launch `wavedump`. The following is an example of a one-shot program that reads `PlotData.txt`, creates a `TH1F` and displays it. This can be run as a root macro, i.e.,

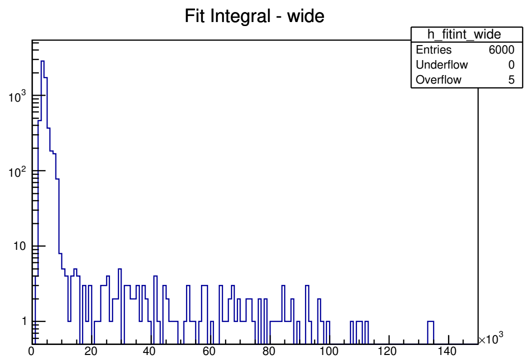
```
root [1] .x NaIDisplay.C
```

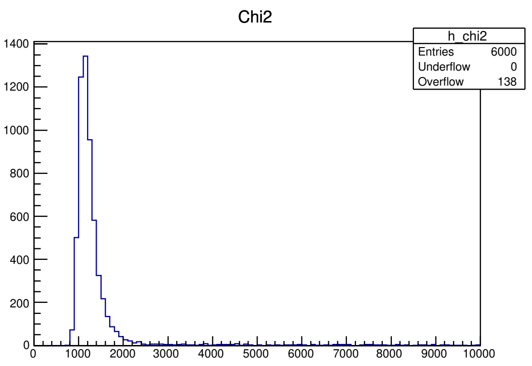
```
void NaIDisplay1()
{
  Int_t debug=1;
  if(debug!=0) printf("NaIDisplay1 - starting\n");

  TGraph* g=new TGraph("/tmp/PlotData.txt");
  auto nPoints=g->GetN();
  printf("nPoints %d\n",nPoints);
  TH1F* h=new TH1F("h","",nPoints,1.,4096.);
  for(Int_t i=0;i<nPoints;++i){
    double x,y;
    g->GetPoint(i,x,y);
    h->SetBinContent(i,y);
```

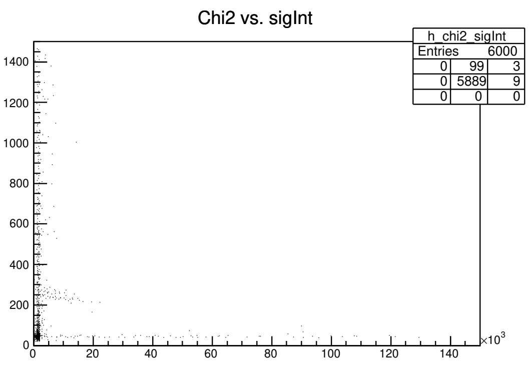

Peak value
h_peak
Entries 6000
Underflow 0
Overflow 148

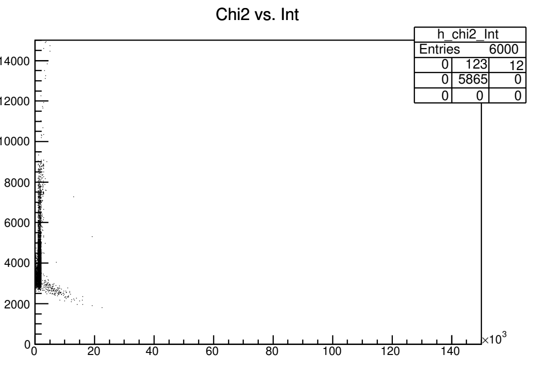

Integral
h_int
Entries 6000
Underflow 0
Overflow 133

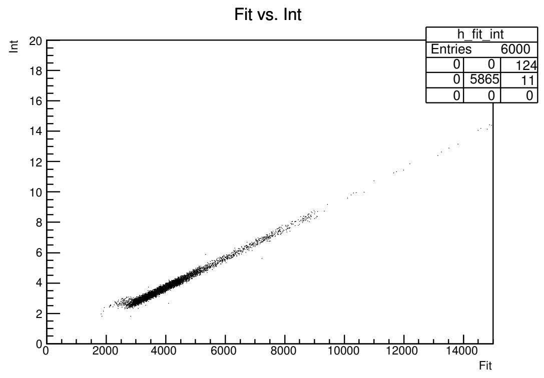

Fit Integral
h_fitint
Entries 6000
Underflow 0
Overflow 135


Fit Integral - wide
h_fitint_wide
Entries 6000
Underflow 0
Overflow 5


Chi2
h_chi2
Entries 6000
Underflow 0
Overflow 138


Chi2 vs. sigInt
h_chi2_sigInt
Entries 6000
0 99 3
0 5889 9
0 0 0


Chi2 vs. Int
h_chi2_Int
Entries 6000
0 123 12
0 5865 0
0 0 0


Fit vs. Int
h_fit_int
Entries 6000
0 0 124
0 5865 11
0 0 0
Int
Fit

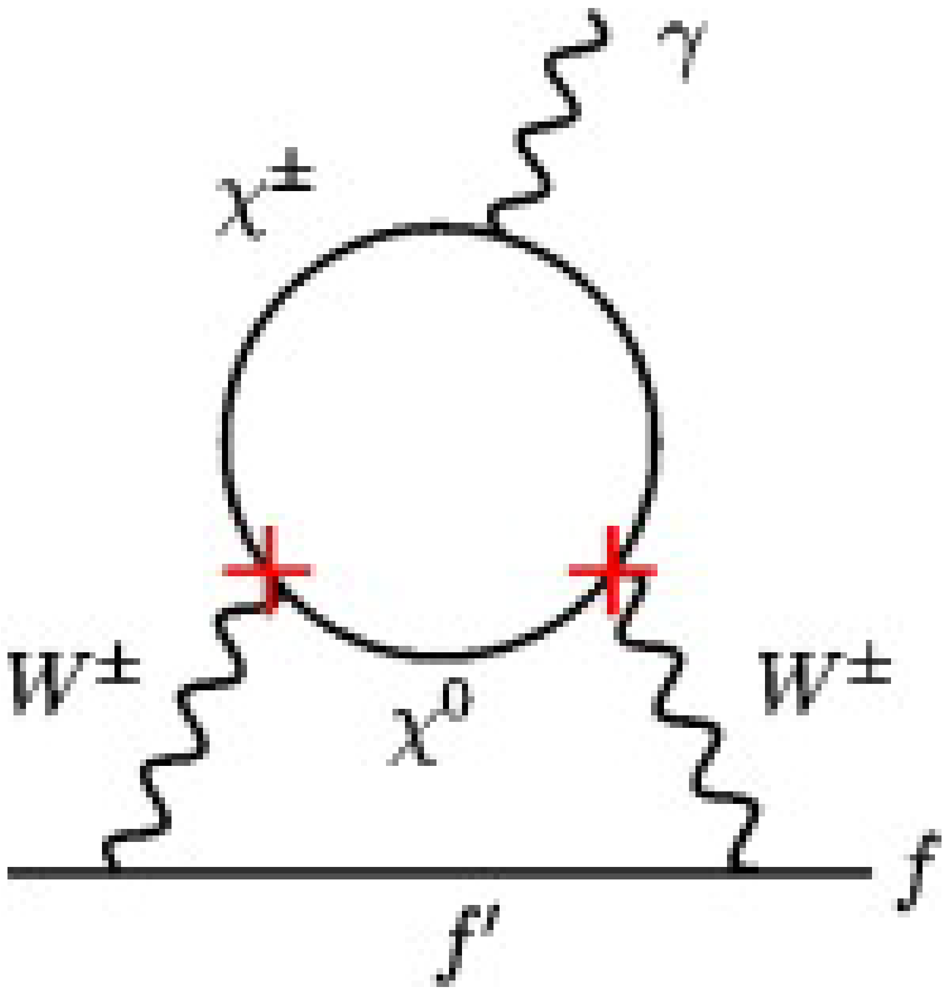


Figure 5: Fit to 2614 keV line for 10 min of live-time.

```
  }
  if(debug!=0) printf("NaIDisplay1 - drawing\n");
  TCanvas* myCan=new TCanvas("myCan","myCan");
  myCan->SetLogy();
  h->Rebin(2);
  h->GetXaxis()->SetRange(1950,2049);
  h->Draw("hist");
}
```

6 Improving Q extraction

The current version of `wavedump` uses peak determination as an estimator for Q. The section above indicates this will lead for a 5% degradation of resolution, making the beam energy determination more difficult. From the argument above, it looks like integration under the peak with a baseline estimate will give adequate results. To move forward quickly, we should replace the peak finder in `wavedump` with a call the a routine with a switch that will allow peak finding, integration, and fitting, with integration at the default. Time permitting, testing to see if there is any advantage to the peak fitting can take place, but as a second priority to ensuring the system works properly with the integration.

The function call should look like,

```
Int_t WaveProcess(char* switch, Int_t* array, Float_t *Q, Float_t *sigmaQ, Float_t

switch = "peak", "int", or "fit"
*Q = total charge collected (in units of 38.4 fC except for "peak" option)
*sigmaQ = uncertainty on *Q
*chi2 = goodness of fit statistic ("fit" only)

WaveProcess should return 1 is the results can be used, 0 if not.
```

Updating the histogram takes place in `WaveDump.c` starting at line 2097.

A test program is in `WaveFit.C` which reads recorded data from the CAEN 6720 in `wave0.txt`. There are 3 M events stored here. The program runs interactively with root, ie.,

```
root [0] .L WaveFit.C++
root [1] fitfunc()
root [2] WaveFit(10000,0)
```

will process 10,000 events, make histograms that will be in `root/WaveForm.root` and plots in `pdf/WaveForm.pdf`. The plots shown in this memo were produced from `WaveForm`.

The immediate task is to extract `WaveProcess` and interface it to `wavedump` for Q extraction. other things that can be done are:

1. Understand if being able to cut events with poor fits improves performance.
2. Is fitting fast enough with `wavedump`?