MEASUREMENT OF BEAM TUNES IN THE TEVATRON USING THE BBQ SYSTEM

Dean R. Edstrom, Jr.

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements for the degree
Master of Science in the Department of Physics, Indiana University
April 2009

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Master of Arts.

Master's Thesis Committee

[redacted]
Dr. Shyh-Yuan Lee, Chairperson

[redacted]
Dr. Cheng-Yang Tan

[redacted]
Dr. Rex Tayloe

[redacted]
Dr. Mike Snow

I would like to acknowledge Cheng-Yang Tan as my advisor at Fermilab and principal reviewer of this thesis. I would also like to acknowledge my committee chair, Shyh-Yuan Lee, and Rex Tayloe & Mike Snow as members of the thesis review committee. Acknowledgement also goes to Marek Gasior of CERN for development of the BBQ System and for distribution to Fermilab for use and analysis. Further acknowledgment belongs to LARP, the USPAS (especially Susan Winchester), and the Indiana University Physics Department (especially Tracy McGooky) for making this Masters Program and thesis possible. Finally, I would like to thank Michael Backfish and the rest of Fermilab's AD/OPS Crew E for reading through sections of this thesis, Bob Mau, John Crawford, and Dan Johnson for support through the IU Masters program, and my family (especially my parents) and friends for general moral support.

Table of Contents

MEASUREMENT OF BEAM TUNES IN THE TEVATRON USING THE BBQ SYSTEM

Dean R. Edstrom, Jr.

Abstract

Measuring the betatron tunes in any synchrotron is of critical importance to ensuring the stability of beam in the synchrotron. The Base Band Tune, or BBQ, measurement system was developed by Marek Gasior of CERN and has been installed at Brookhaven and Fermilab as a part of the LHC Accelerator Research Program, or LARP. The BBQ was installed in the Tevatron to evaluate its effectiveness at reading proton and antiproton tunes at its flattop energy of 980 GeV. The primary objectives of this thesis are to examine the methods used to measure the tune using the BBQ tune measurement system, to incorporate the system into the Fermilab accelerator controls system, ACNET, and to compare the BBQ to existing tune measurement systems in the Tevatron.

Tevatron Tune Measurement

The Tevatron at Fermilab is a synchrotron that accelerates counter rotating beams of protons and antiprotons injected from the Main Injector at 150 GeV to a flattop energy of 980 GeV with each beam consisting of 36 bunches of its respective species separated by means of electrostatic separators. Through a series of lattice adjustments at 980 GeV the beams are brought into contact at two points, called the collision points, in the detectors of the CDF and D0 experiments with a collision-point energy of almost 2 TeV. Beam parameters, such as orbit, betatron tune, and chromaticity, are closely monitored using a number of diagnostics and corrected using dipoles, quadrupoles, sextupoles, and octopoles. The bunches of beam are arranged as shown in *Figure 1*.

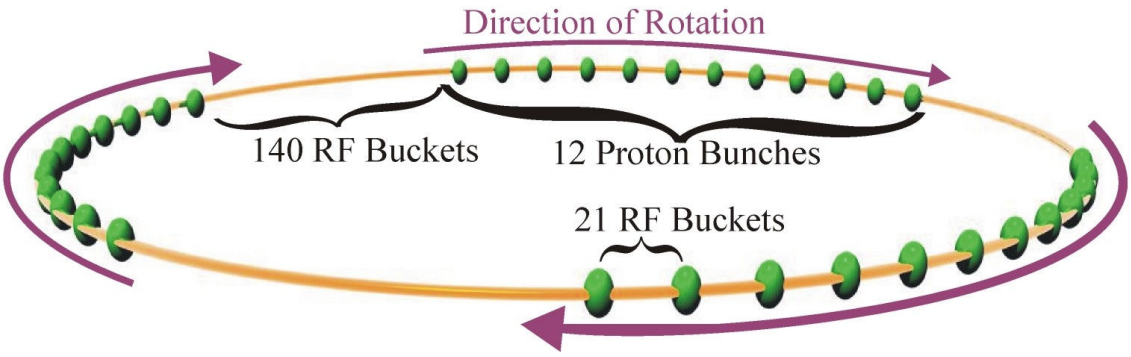

Figure 1 – Shown here is the proton bunch distribution in the Tevatron as viewed from above. The bunches are arranged into three trains of 12 bunches for a total of 36 bunches. The inter-bunch spacing within each train is 21 RF buckets and an inter-train spacing of 140 RF buckets. The proton bunches travel around the Tevatron clockwise, as indicated by the arrows, with an energy between 150GeV at injection and 980GeV at flattop, where collision data is collected by the detector experiments at CDF and D0. The antiproton bunches are arranged in the same manner but rotate counter clockwise in the beam pipe with the two beams separated using electrostatic separators.

The betatron tune is the number of natural oscillations of the beam in the transverse plane about the central orbit arising from the phase advance through a repeating lattice over the period of a single revolution. If these were to align such that they kick the beam resonantly every revolution the amplitude of the oscillations would increase until the beam leaves the beam pipe and is lost. This is referred to as an integer resonance condition. Higher order resonance conditions will occur if the phase advance

brings the beam back to the same point in the oscillation every 2nd, 3rd, or higher ordinal revolution resulting in a net increase in the amplitude of oscillation over many revolutions. In order to prevent the beam from blowing up, the tune is kept away from these integer resonances and stronger, non-integer resonances, such as $^1/_2$, $^1/_3$, $^1/_4$ and so on, using strings of quadrupoles.[i,ii] The typical full betatron tunes of protons and antiprotons in the Tevatron usually reside in a stable area in tune-space between 20.575 and 20.595. Because the integer part of the tune remains fixed, it is typically omitted from recorded tune values leaving only the fractional tune. For example, the above tune space would be recorded as 0.575 to 0.595 in most documentation, as shown in *Figure 2*. Unless otherwise stated, only the fractional tune will be used for the remainder of this paper.

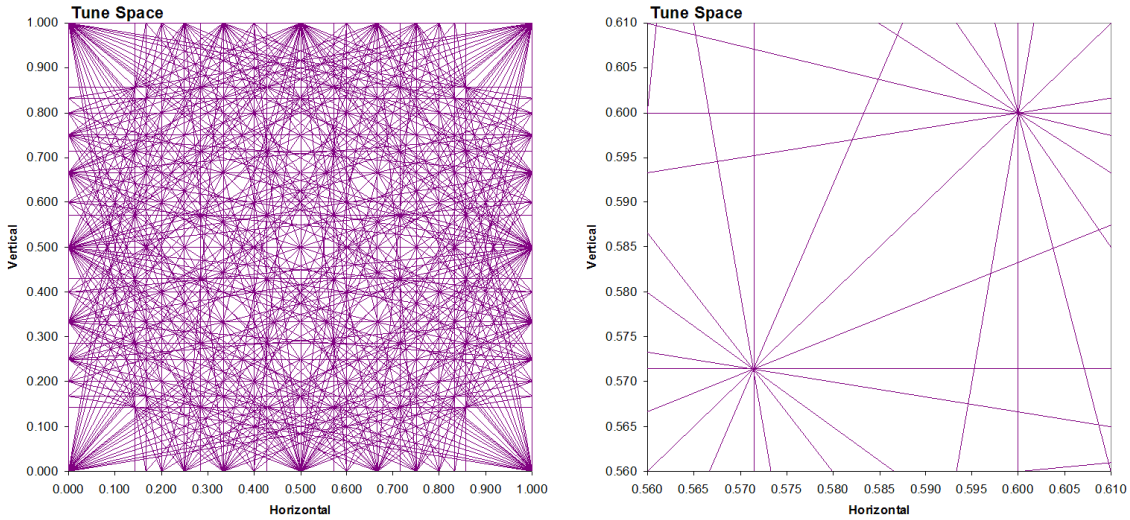

Figure 2 – Tune-space resonance plots showing resonances up to 7th order. The plot on the left shows the entire harmonic tune space while the plot on the right shows a detail the typical tune space of the tunes in the Tevatron, from 0.575 to 0.595 in each plane. The lines on the plot represent potential points of instability with lower order resonances being most likely to result in an instability.

Measurement of the tunes begins with detection by pairs of transverse beam pickups. The signals from these pickups, $s_e^+(t)$ and $s_e^-(t)$, are shown in *Figure 3* and can carry with them a wealth of information such as the position, tune and chromaticity of the beam. $s_c(t)$ is the difference between the two pickups for the plane in question:

$$s_c(t) = s_e^+(t) - s_e^-(t) \tag{1}$$

Measurement of $s_c(t)$ can be performed with an oscilloscope or other device that allows current flow through a known resistance. It can therefore be thought of as either a time-dependant voltage or current signal. Converting from one to the other only requires knowing the input impedance of the measurement device, which is typically 50Ω or 1MΩ. Since oscilloscopes typically display in units of volts, the time-dependant voltage signals, $s_x(t)$, are understood to be in units of volts.

Given a coasting, DC beam with a constant transverse offset from the center of the pickups, and undergoing simple sinusoidal betatron motion with no chromaticity or momentum spread, the beam position at the pickups will oscillate with the betatron frequency, f_q. As a result the voltage on each of the plates will oscillate as well, reaching extrema when the beam is closest to the pickups as shown in *Figure 4*. The time dependant signal for this can be written as

$$s_{cDC}(t) = A_{oDC} + A_{bDC} \cdot \cos(2\pi f_q t) \tag{2}$$

where A_{oDC} comes from the closed orbit of the beam, A_{bDC} comes from the amplitude of the betatron oscillation, and $s_{cDC}(t)$ is the difference between the terminals.

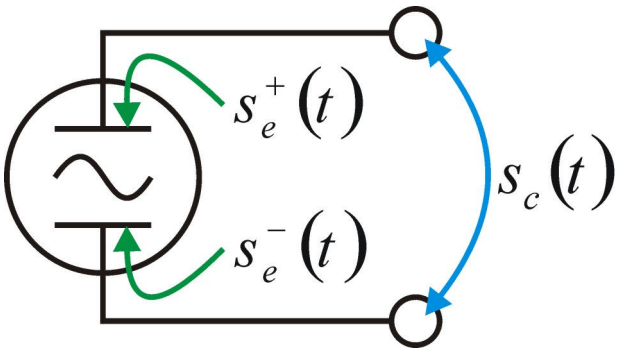

Figure 3 – A Basic Pickup Diagram.

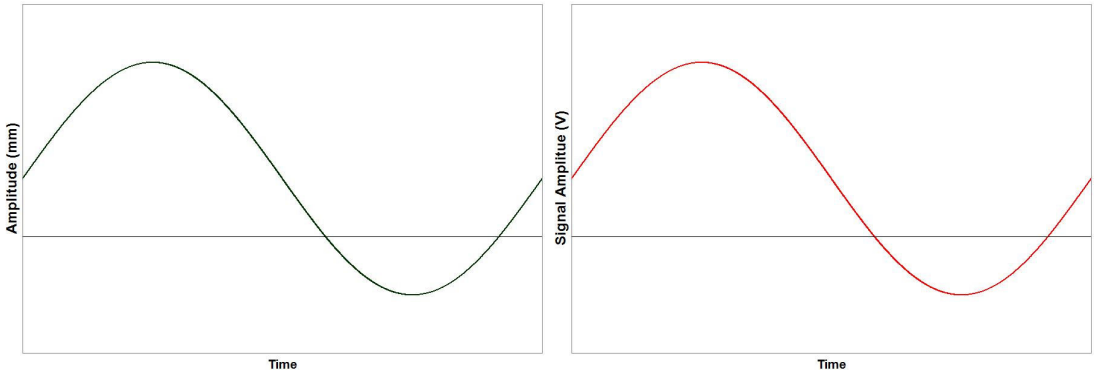

Figure 4 – The motion of a beam of charged particles as it passes a pair of pickups, shown to the left, results in a voltage variation that tracks the position of the beam resulting in a time-dependant potential signal, $s_c(t)$, shown to the right.

If a single point charge is considered instead, the bunch only passes the pickup once per revolution period. As a result the pickup only samples the sinusoidal betatron oscillations at those points when the beam is present. This can be expressed as a series of Dirac delta functions,

$$\delta_T(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{3}$$

where T is the revolution period of the particle around the Tevatron and n is the revolution number, which includes all previous and future revolutions. The resulting signal, shown in *Figure 5*, can be expressed as

$$s_{cI}(t) = \left[A_{oI} \cdot \delta_T(t)\right] + \left[A_{bI} \cdot \cos(2\pi f_q t) \cdot \delta_T(t)\right] \tag{4}$$

Where A_{oI} is the central-orbit amplitude of the impulse and A_{bI} is the betatron impulse oscillation amplitude. In general these amplitude potentials will not be the same as A_{oDC} and A_{bDC}. Assuming that the same amount of beam is used, the summed area under the signals should be the same over a single revolution period. Consequently they should also be the same over a single betatron oscillation period as well. For the DC beam shown above with frequency of $f_q = 1/T_q$, as seen in *Figure 4*, the area under the curve for a half-period is

$$\left.\begin{aligned}\int_{-T_q/4}^{T_q/4}\left[A_{oDC}+A_{bDC}\cdot\cos\left(2\pi f_q t\right)\right]dt &= \frac{A_{oDC}T_q}{2}+\frac{A_{bDC}T_q}{2\pi}\left[\sin\left(\frac{2\pi t}{T_q}\right)\right]\Bigg|_{-T_q/4}^{T_q/4}\\ &= \left(\frac{A_{oDC}}{2}+\frac{A_{bDC}}{\pi}\right)\cdot T_q\end{aligned}\right\} \quad (5)$$

in units of Volt-Seconds, or Webers. For true point charges, this would result in infinite amplitudes since the Dirac function has no expanse and therefore no area under which to integrate.

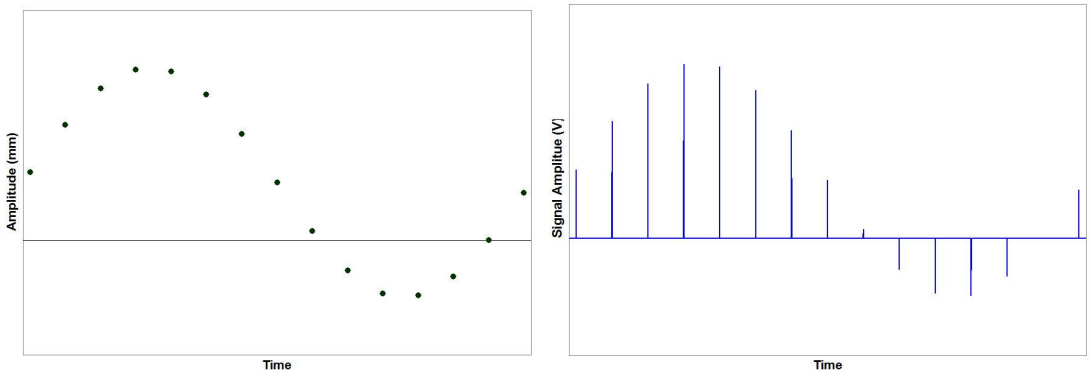

Figure 5 – The point charge passing the pickups will result in a series of Dirac spikes on an oscilloscope. The spikes are modulated by the sinusoidal betatron function and offset of the central-orbit proportionally to the physical offset of the beam.

In practice the beam will have a distribution, which will result in a spreading-out of the beam signal. Mathematically this is expressed as a convolution between the betatron component from *Equation 4*, $\cos\left(2\pi f_q t\right)\cdot\delta_T(t)$, and a bunch distribution function, $s_b(t)$. The closed orbit component, $s_o(t)$, will be a function of time as well to appropriately offset the entire bunch signal while allowing the signal to return to 0 between bunches. For a constant offset of the beam, this can be described as a Heaviside step function, $H(t)$, up at the leading edge of the bunch, constant during the bunch and a second Heaviside step function back to 0 at the trailing edge at time t_0, $H(t-t_0)$ where,

$$H(t) = \int_{-\infty}^{t} \delta(x)dx \tag{6}$$

as shown in *Figure 6*.

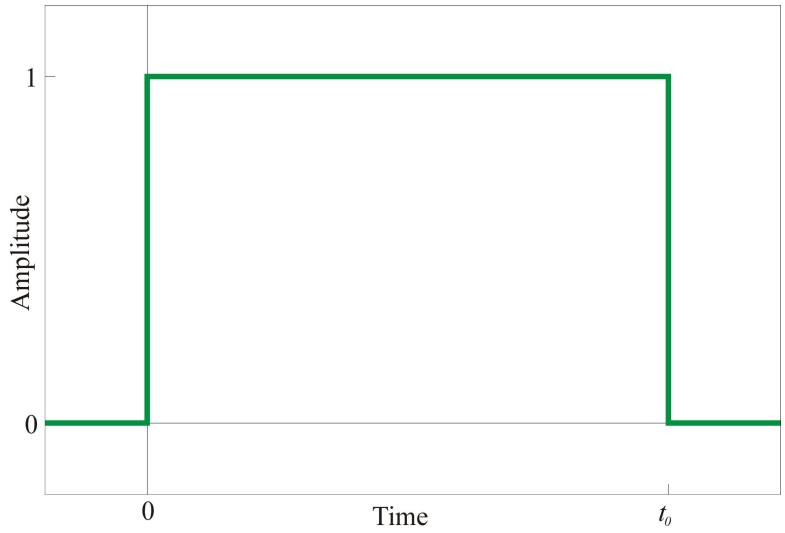

Figure 6 – A Heaviside step function pair or 'boxcar function.'

The Heaviside function pair is referred to as a Boxcar function,

$$\prod_{t_1,t_2}(t) = H(t-t_1) - H(t-t_2) \tag{7}$$

To line up the Heaviside step function pair to be centered on the Dirac delta function it must be offset by ½ of its width. The offset signal would just be a scaled Heaviside step function pair with an expanse of t_0,

$$s_o(t) = A_o \cdot \prod_{\frac{-t_0}{2},\frac{t_0}{2}}(t) \tag{8}$$

As with the bunch distribution function, this closed-orbit function is convolved with the Dirac delta such that

$$s_c(t) = [s_o(t) * \delta_T(t)] + \cos(2\pi f_q t)[s_b(t) * \delta_T(t)] \tag{9}$$

where the convolution of two functions, f and g, is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t-\tau) d\tau = \int_{-\infty}^{\infty} f(t-\tau) \cdot g(\tau) d\tau \tag{10}$$

by symmetry. This superimposes the given signals on to the delta function as shown in *Figure 7*.

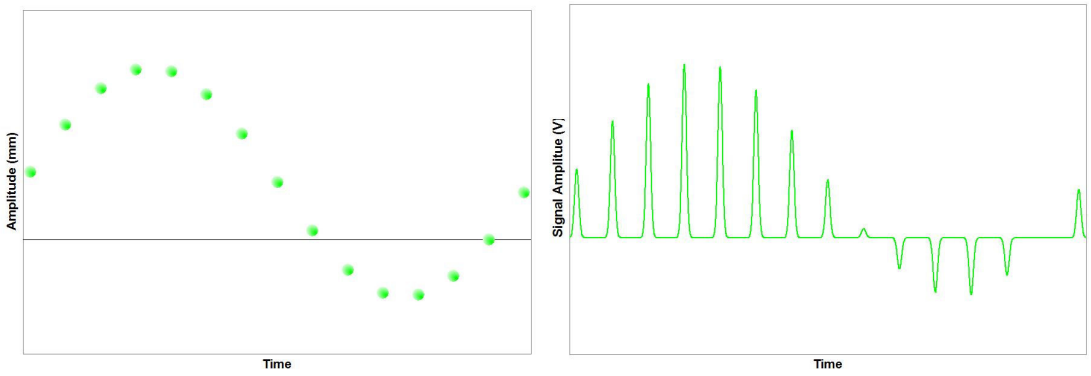

Figure 7 – Beam with a distribution results in a convolution between the beam distribution and offset functions and the periodic Dirac spikes.

As with the point-charge case in *Figure 5*, the sum of the area under the curves in *Figure 7* should have the same area, in Webers, as the DC beam in *Equation 5*. This integration, however, depends on the time-dependant bunch and offset distributions, which can be quite complicated depending on the shape of the beam distribution and offset. As a result, it is not easily approached analytically.

Simply stated, measurement of the tune at this point is merely a question of calculating the frequency of the sinusoidal envelope, f_q. In practice there are several additional complicating factors, such as momentum spread, impulse response of the pickups, and background noise that distort the clean signal seen in *Figure 7*, resulting in a time-dependant terminal voltage signal, $s_c(t)$, that makes calculation of the tune difficult. It is far easier to examine this in the frequency domain which requires a Fourier transform,

$$S(f) = \mathcal{F}\{s(t)\} = \int_{-\infty}^{\infty} s(t) e^{-2\pi i f t} dt \tag{11}$$

where $S(f)$ is the frequency spectrum of $s(t)$ and j denotes an imaginary number, such that $j^2 = -1$. The spectrum area is generally measured in Volts or as a function of power, in dBm, per unit frequency on a vector signal analyzer or similar instrument. From *Equation 10*, applying a Fourier transform to a convolution of two functions, such as the one in *Equation 9*, results in the product of the Fourier transform of each constituent function. Applying a Fourier transform to a delta function, one finds

$$\mathcal{F}\{\delta(t-nT)\} = e^{-2\pi jfnT} \tag{12}$$

which will only have a real solution for $f = 1/T$, resulting in a Dirac delta. For an infinite sum over all integers, n, this results in a sum of delta functions in frequency space. Finally, a sinusoidal function can be considered in terms of the trigonometric identities,

$$\left.\begin{aligned} \cos(x) &= \Re\left(e^{jx}\right) = \frac{e^{jx}+e^{-jx}}{2} \\ \sin(x) &= \Im\left(e^{jx}\right) = \frac{e^{jx}-e^{-jx}}{2j} \end{aligned}\right\} \tag{13}$$

and will result in delta spikes at harmonics of the fundamental frequency. If the sinusoidal function should have an envelope, the Fourier transform of the envelope function will be repeated at the fundamental frequency and higher harmonics. These identities can be summarized as

$$\left.\begin{aligned} &\mathcal{F}\{a(t) * b(t)\} = A(f)\cdot B(f) \\ &\mathcal{F}\left\{\sum_{n=0}^{\infty}\delta(t-nT)\right\} = \sum_{n=0}^{\infty}\delta\left(f-\frac{n}{T}\right) \\ &\mathcal{F}\{a(t)\cdot\cos(2\pi f_0 t)\} = \tfrac{1}{2}\left[A(f-f_0)+A(f+f_0)\right] \\ &\mathcal{F}\{1\} = \delta(f) \end{aligned}\right\} \tag{14}$$

Applying these to *Equation 9*, one will find that the spectrum of $s_c(t)$ can be expressed as

$$\left.\begin{aligned} S_c(f) &= \left|\mathcal{F}\{s_c(t)\}\right| \\ &= \left|\tfrac{1}{2}\left[\sum_n S_b\left(f - f_q - \tfrac{n}{T}\right) + \sum_n S_b\left(f + f_q - \tfrac{n}{T}\right)\right]\right. \\ &\quad \left. + \left[\sum_n S_o\left(f - \tfrac{n}{T}\right)\right]\right| \end{aligned}\right\} \qquad (15)$$

where the expressions $S_b(f)$ and $S_o(f)$ are the Fourier transforms of the bunch distribution and closed-orbit functions. Additional envelopes may affect the measured signal, such as the response of the pickups, $P(f)$. In the time domain the time function of such a response, $p(t)$, would be convolved with *Equation 9*, but in the frequency domain the response function can be factored in directly such that

$$\left.\begin{aligned} S_c(f) &= \left|\tfrac{1}{2}\left[P\left(f - f_q\right)\cdot\sum_n S_b\left(f - f_q - \tfrac{n}{T}\right)\right.\right. \\ &\quad \left.\left. + P\left(f + f_q\right)\cdot\sum_n S_b\left(f + f_q - \tfrac{n}{T}\right)\right]\right. \\ &\quad \left. + \left[P(f)\cdot\sum_n S_o\left(f - \tfrac{n}{T}\right)\right]\right| \end{aligned}\right\} \qquad (16)$$

which effectively confines the spectra from *Equation 15* to an envelope. One can consider *Equations 15* and *16* to be the same given perfect pickups, such that $P(f) = 1$ for all frequencies *f*. Other system frequency responses can be accounted for in the same manner. *Equation 16* can be thought of as three distinct periodic segments,

$$S_c(f) = \left|\tfrac{1}{2}S_{SB}^{-} + \tfrac{1}{2}S_{SB}^{+} + S_{CM}\right| \qquad (17)$$

Where

$$S_{SB}^{\pm} = \tfrac{1}{2} P\left(f \pm f_q\right) \cdot \sum_n S_b\left(f \pm f_q - \tfrac{n}{T}\right) \tag{18}$$

are the upper and lower tune sidebands to the common mode, revolution frequency harmonics,

$$S_{CM} = P(f) \cdot \sum_n S_o\left(f - \tfrac{n}{T}\right) \tag{19}$$

Unless deliberately made otherwise, the distribution of protons bunched in sinusoidal RF buckets can usually be approximated as Gaussian. A Gaussian bunch distribution will have a time-domain structure of

$$s_g(t) = A_g \cdot e^{-t^2/2\sigma^2} \tag{20}$$

where σ is the standard deviation of the Gaussian and A_g is the Gaussian amplitude. The voltages on the pickups, $s_e^+(t)$ and $s_e^-(t)$, will be Gaussian and, as a result, so will the bunch intensity response and orbit, $s_b(t)$ and $s_o(t)$. In the frequency domain, this means that the bunch intensity response and orbit spectra will follow the Gaussian spectrum,

$$\left.\begin{aligned} S_g(f) &= \left|\mathcal{F}\left\{s_g(t)\right\}\right| \\ &= \left|A_g \cdot \int_{-\infty}^{\infty} e^{-t^2/2\sigma^2} \cdot \left[\cos(2\pi f t) + j\sin(2\pi f t)\right] dt\right| \\ &= \sqrt{2\pi} A_g \sigma e^{-2\pi^2\sigma^2 f^2} \end{aligned}\right\} \tag{21}$$

One will note that the exponential has a factor of the frequency in it that leads to degradation and eventual extinction of the common modes and sidebands, at higher harmonics as shown *Figure 8*.[iii]

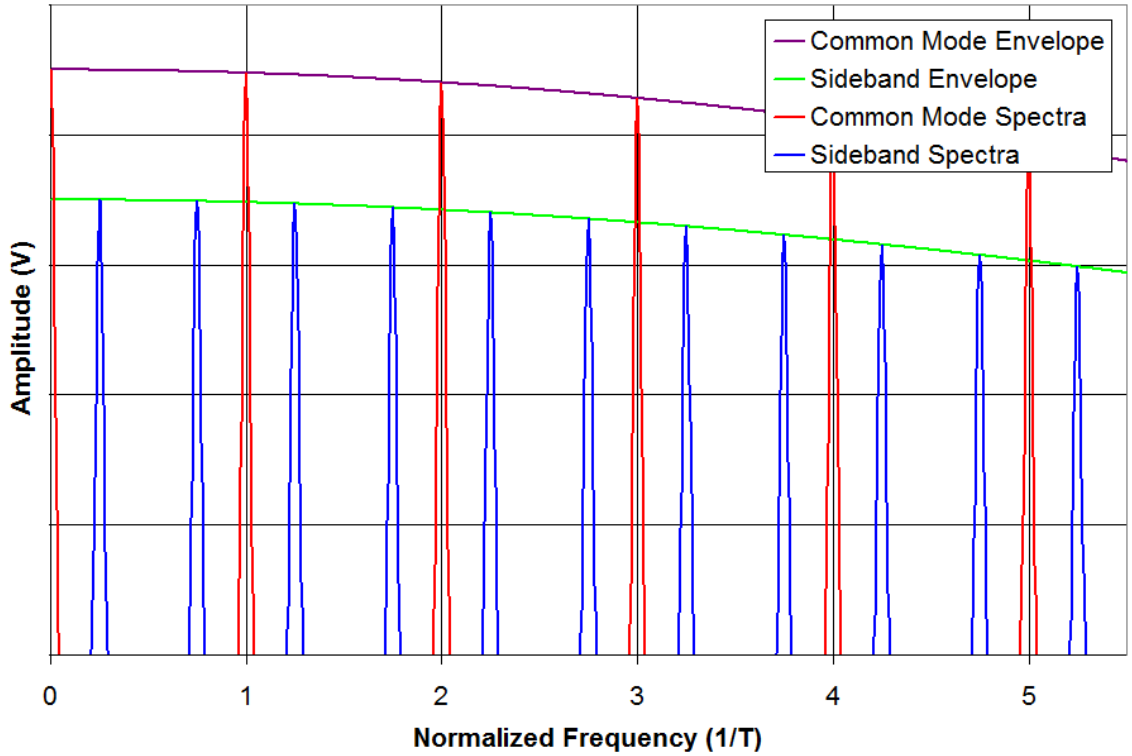

Figure 8 – A progression of the revolution frequencies and their corresponding betatron side-bands over the first few harmonics.

As seen from the periodic term in *Equation 18*, the sidebands occur at $\delta\left(f \pm f_q - \frac{1}{T}\right)$ in the baseband. Considering, for a moment, a tune of exactly 0.000, the tune sidebands, at $f = f_q$, could be thought of as overlapping the common mode. Increasing the tune from 0.000 to 1.000, the sidebands would spread apart from the common mode until they overlap the next harmonic. The upper sideband therefore moves up in frequency space and the lower sideband moves down in frequency space with increasing tune such that

$$\pm q^{\pm} = \left(f_{q^{\pm}} T\right) - 1 \tag{22}$$

in the baseband, where q^+ and q^- represent the upper and lower sidebands respectively. The common mode will be at the revolution frequency, f_r, such that

$$f_r = 1/T = f_{RF}/h \tag{23}$$

Where f_{RF} is the RF frequency and h is the harmonic number, or the number of RF oscillations in one revolution period. The frequency difference between the tune sidebands and their common mode will remain the same for higher harmonics of f_r, and therefore the tune will remain the same as well. In the Tevatron, the harmonic number is 1113 and the RF frequency at the collision energy of 980 GeV is about 53.1047 MHz, which results in a single-bunch revolution frequency of f_r = 47.713 kHz according to *Equation 23*. The tunes can then be found by moving off the revolution frequency fundamental by a fraction of that frequency. The Tevatron tunes are maintained at around 0.590 at collisions. The corresponding frequencies of these sidebands are 47.713 kHz ± 28.151 kHz at 980 GeV for the lower and upper sidebands respectively according to *Equation* 22.

Ideally, the tunes would be fixed during the ramp from 150 GeV to 980 GeV to prevent their approaching a resonance, but there are a number of factors that make this unrealistic. Even with a fixed desired tune, q, the frequency sidebands will change between the injection energy of 150 GeV and the collision energy of 980 GeV due to dependence on the RF frequency. In principle, this would have to be taken into account when converting measurements from frequency to tune space. From *Equations 22* and *23*,

$$\Delta f_q = \left(\frac{1 - q^-}{h} \right) \Delta f_{RF} \tag{24}$$

for the lower tune sideband. For example, a desired tune of 0.590 will result in a lower sideband frequency shift of about Δf_q = 0.371 Hz given h and the RF frequencies mentioned above. As will be seen later, this is very small compared to not only the sideband frequency, f_q, but the sideband width as well. It can therefore be considered negligible, and the sideband frequency remains relatively constant at f_q = 19.562 kHz.

The change in the RF frequency, however, is not the only point one must consider. The quadrupole fields, for example, must change through the ramp from 150

GeV to 980 GeV resulting in highly energy-dependant tunes. The bunch length will also change through the ramp, which will change the momentum spectrum from *Equation 21*, as shown in *Figure 9*.

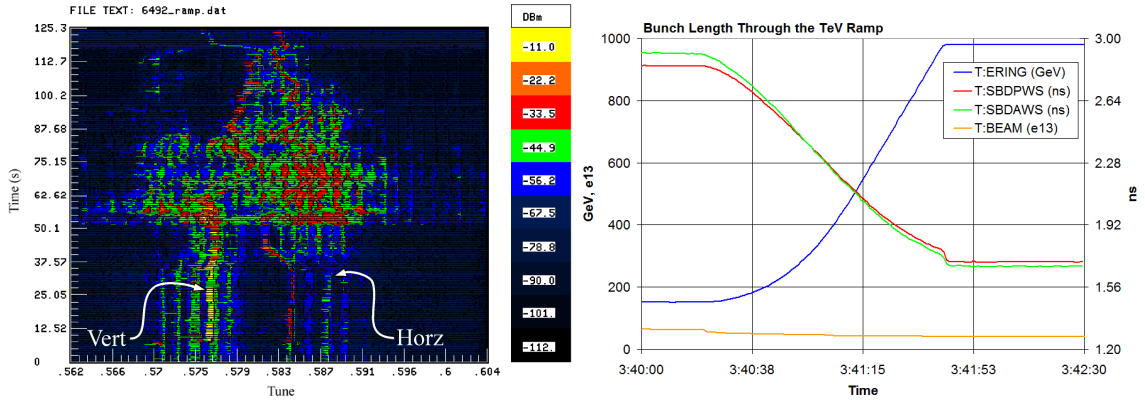

Figure 9 – Horizontal and Vertical tune spectra from the 21.4MHz Schottky (left) and proton & antiproton bunch lengths, T:SBDPWS and T:SBDAWS respectively, through the ramp from 150GeV to 980GeV (right). The beam energy, T:ERING, and total number of particles, T:BEAM, are also shown.

There are currently four different tune measurement systems in the Tevatron: the 1.7GHz Schottky, the 21.4MHz Schottky, the BBQ system, and the Digital Tune Monitor. This paper will primarily focus on comparing results from the Direct Diode Detector Baseband Tune, or 3D-BBQ, system and the 1.7GHz Schottky as a representative operational tune measurement. While the BBQ and 1.7GHz systems look at different harmonics, the baseband and 35630th harmonic respectively, the tune information is still comparable since only the amplitude, and not the relative frequency, of the tune sidebands change with the common mode harmonic. The primary motivation for using the BBQ system is due to the difficulty of monitoring the tunes at flattop due to coherent effects on the beam. These effects are present starting at the beginning of High Energy Physics, or HEP, at flattop and raise the noise floor making tune measurement problematic. They are not currently well understood.[iv] Furthermore, the BBQ system is designed to measure tunes using the intrinsic betatron motion, on the order of nanometers, without the aid of pinging to excite oscillatory motion.

The 3D-BBQ System

Tune measurements with the BBQ system differ from traditional tune measurements because of the use of Direct Diode Detectors and BBQ front end, shown in *Figure 10.* As the beam passes the pickups, the resulting signal goes through a transmission line to the E0 service building. The pickups used are directionally coupled, such that the signal from particles traveling one direction, the protons for example, is much greater than the signal from particles traveling in the opposite direction, the antiprotons in this case. Once in the E0 service building the signals are routed through the Direct Diode Detectors, which are diode boxes, to the BBQ front end, where the DC offset is removed and the difference of the signals is taken. The front end then applies user-specified filtering and amplification to the signal before sending it on to a Vector Signal Analyzer, or VSA.

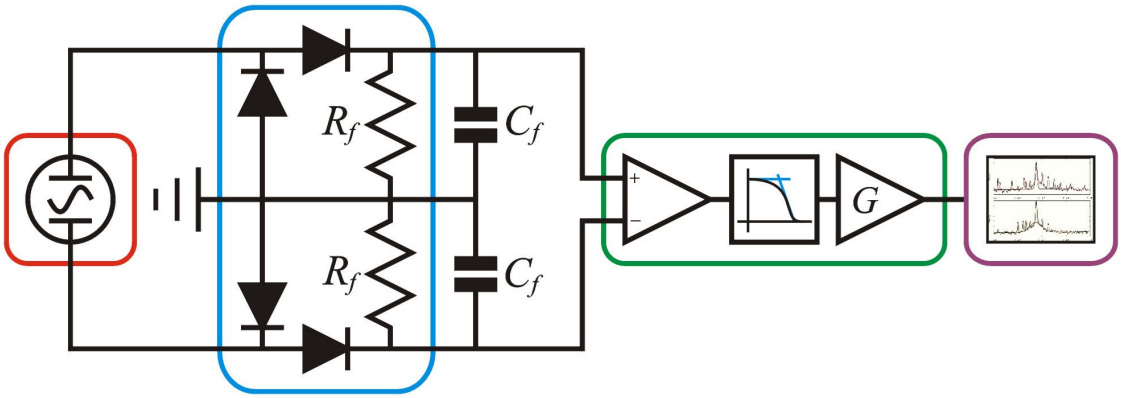

Figure 10- The basic circuit diagram of the BBQ setup. The main pieces of the BBQ setup are shown in the colored boxes. The pickups, in the tunnel, are in the red box, the diode detectors are in the blue box, the BBQ front end is in the green box, and the VSA is in the purple box. The BBQ Front End consists of a high-impedance differential amplifier followed by user-selectable filtering and gain amplifiers

Because the signal changes dramatically from the pickups to the VSA, as described in the BBQ design report, it is instructive to examine the signal at each point along its path. Starting with the pickups, the signals were examined using a Tektronix

TDS 644B oscilloscope. Clear doublets were observed for the target species of particle in each plane, and very small doublets could also be seen because the opposite species were not fully cancelled as seen in *Figure 11*.

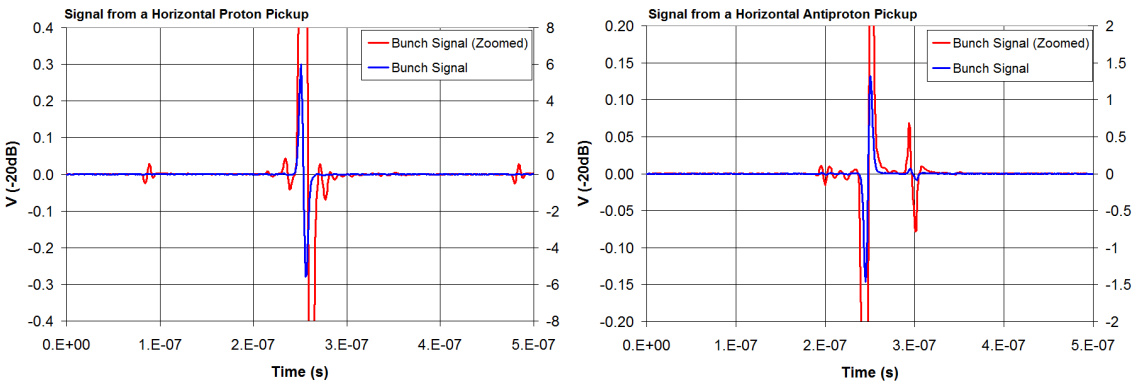

Figure 11 – A signal sample from one of the proton horizontal pickups is shown on the left. It has a strong proton signal doublet (blue) and the same signal on a tighter amplitude scale to reveal heavily-attenuated antiproton signal doublets (red) on either side. The two traces are from two separate measurements and therefore aligned by the scope trigger, which is locked to the leading edge of the doublet. The signals from the antiproton vertical pickups, on the right, are similar, but weaker due to the lower intensity of the antiproton bunches. The doublet amplitudes are inverted due to the opposite charge of the antiprotons from that of the protons. As a result, the proton bunch can be more clearly seen to the right of the antiproton bunch, but is still heavily attenuated compared to the amplitude of the antiproton signal.

The Direct Diode Detectors are peak detectors such that the stored potential in the shunt capacitor, C_f, jumps up in response to the voltage amplitude on the pickup and then decays away through the shunt resistor, R_f, with the canonical time constant for an RC circuit,

$$\tau = R_f \cdot C_f \tag{25}$$

The resulting signal can be modeled as a Heaviside step function, $H(t)$ as described in *Equation 6*, up at time $t = 0$ followed by decay. Considering a single revolution period, an additional Heaviside step function back to the baseline, $H(t-T)$, is subtracted resulting in the signal model

$$s_f(t) = e^{-t/\tau} \cdot \prod_{0,T}(t) \tag{26}$$

referred to as the normalized filter hold function, shown in *Figure 12*.

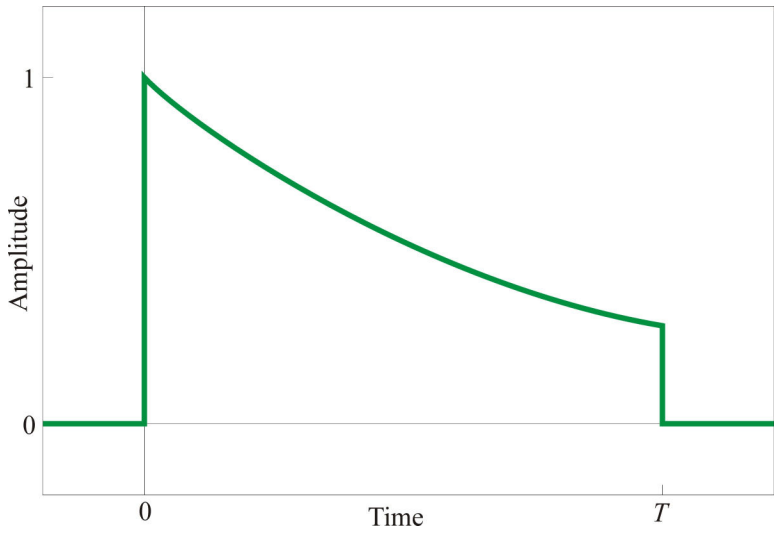

Figure 12 – The normalized filter hold function

Taking the Fourier transform of this model, the complex spectrum is found to be

$$S_f(f) = \int_0^T e^{-2\pi jft - \frac{t}{\tau}} dt = \tau\left(\frac{1 - e^{-\left(2\pi jfT + \frac{T}{\tau}\right)}}{1 + 2\pi jfT}\right) \tag{27}$$

which simplifies considerably for a large normalized time constant, τ/T, such that the decay can be ignored, allowing $s_f(t) \approx 1$ between times t = 0 and t = T. The resulting complex spectrum is

$$\left.\begin{aligned} S_f(f) &= \int_{-\infty}^{\infty} s_f(t) e^{-2\pi jft} dt \approx \int_0^T e^{-2\pi jft} dt \\ &= -\tfrac{1}{2\pi jf} e^{-2\pi jft}\Big|_0^T = \tfrac{1}{2\pi jf}\left(e^{\pi jfT} - e^{-\pi jfT}\right) e^{-\pi jfT} \end{aligned}\right\} \tag{28}$$

Taking the modulus of this complex spectrum results in the normalized filter hold function spectrum,

$$\left|S_f(f)\right| = \left|\tfrac{1}{\pi f} \cdot \sin(\pi f T)\right| \tag{29}$$

Unlike the ideal differential signal from the standard approach in *Equation 9*, the signal after the detector in the time domain will spike in response to beam passing a pickup and the signal will decay away with a time constant according to *Equation 25*. Both the betatron oscillation and the central-orbit amplitudes contribute to the amplitude of this spike. This results in the detector output,

$$s_d(t) = A_b \cdot \cos(2\pi f_b t)\left[s_f(t) * \sum_n \delta(t - nT)\right] + A_o\left[s_f(t) * \sum_n \delta(t - nT)\right] \tag{30}$$

where $s_f(t)$ is the filter hold function from *Equation 26*, A_b is the betatron oscillation scale factor and A_o is the orbit scale factor. Using the identities in *Equations 14*, the spectrum of the detector output is found to be

$$\left.\begin{aligned} S_d(f) &= \left|\mathcal{F}\{s_d(t)\}\right| \\ &= \left|\tfrac{1}{2}\left[A_b \cdot \sum_n S_f\left(f - f_q - \tfrac{n}{T}\right) + A_b \cdot \sum_n S_f\left(f + f_q - \tfrac{n}{T}\right)\right]\right. \\ &\quad \left. + \left[A_o \cdot \sum_n S_f\left(f - \tfrac{n}{T}\right)\right]\right| \end{aligned}\right\} \tag{31}$$

where $S_f(f)$, is the Fourier transform of the filter hold function from *Equation 29* at frequency *f*. This, again, is composed of the three components as shown in *Equation 17*, where

$$S_{SB}^{\pm} = \tfrac{1}{2} A_b \cdot \sum_n S_f\left(f \pm f_q - \tfrac{n}{T}\right) \tag{32}$$

are the upper and lower tune sidebands to the common mode harmonics

$$S_{CM} = A_o \cdot \sum_n S_f\left(f - \tfrac{n}{T}\right) \tag{33}$$

Substituting in the expression for the normalized filter hold function spectrum from *Equation 29* into the sideband spectrum components from *Equation 32* the sideband spectra for long time constant, τ, can be approximated as

$$\left.\begin{aligned} S_{SB}^{\pm} &\approx \tfrac{1}{2} A_b \cdot \sum_n \left| \frac{\sin\left[\pi T \cdot \left(f \pm f_q - \tfrac{n}{T}\right)\right]}{\pi \cdot \left(f \pm f_q - \tfrac{n}{T}\right)} \right| \\ &= \tfrac{T}{2} A_b \cdot \sum_n \left| \operatorname{sinc}\left[T \cdot \left(f \pm f_q - \tfrac{n}{T}\right)\right] \right| \end{aligned}\right\} \tag{34}$$

by symmetry. Due to the $^1/_f$ dependence on the amplitude of the sinc function, the majority of the signal resides in the first few harmonics. The same relationship of the tune frequencies to the common mode applies as stated in *Equation 22* and the result is that the tunes reported by the 3D-BBQ system are the same as traditional tune measurement systems. Similarly, the common mode signal can also be approximated to be

$$S_{CM} = T \cdot A_o \sum_n \left| \operatorname{sinc}\left[T \cdot \left(f - \tfrac{n}{T}\right)\right] \right| \tag{35}$$

In the interest of maximizing the betatron signal, it is desirable to minimize the common mode signal. To this end one may define a degree of suppression, ξ, as the ratio between the common mode signal, S_{CM}, at the tune frequency, f_q, and the revolution frequency, f_r. As a first-order approximation, the tune frequency will be relatively close to the half-integer resonance as compared to the revolution frequency.[v] The corresponding degree of suppression can be expressed as

$$\xi = \frac{S_f\left(\frac{f_r}{2}\right)}{S_f\left(f_r\right)} = \sqrt{4 - \frac{3T^2}{T^2 + \pi^2\tau^2}} \coth\left(\frac{T}{2\tau}\right) \tag{36}$$

This common mode minimization can be seen when plotting *Equation 31* after substituting in *Equations 34* and *35* as shown in *Figure 13*.

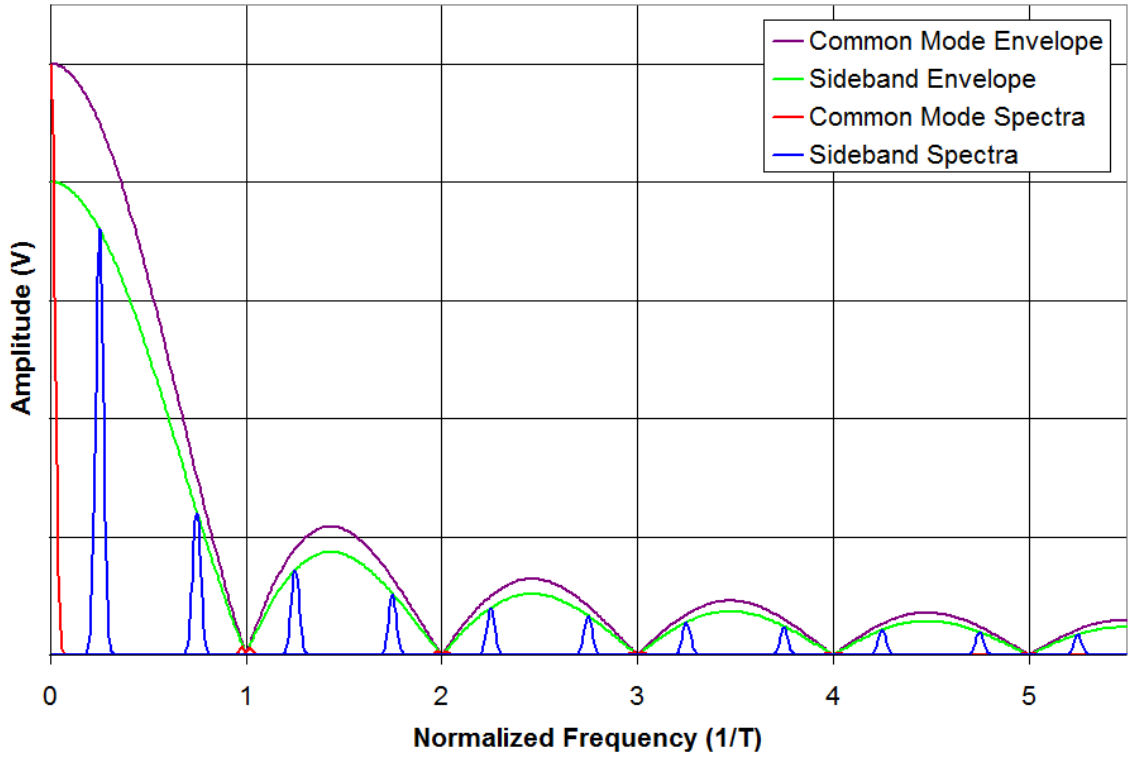

Figure 13 – The spectrum described by *Equation 31* and subsequent analysis for high time constant, τ. This plot is analogous to the standard spectrum shown in *Figure 8*. Note that the envelope is a minimum at harmonics of the revolution frequency, resulting in suppression of the revolution frequency common mode.

There were four different varieties of diode box provided for use with the BBQ front end. These were the "Protons", "Pbars", "A Antiprotons", and "B Protons" as shown in *Figure 14*. The only difference between the "Protons" and "Pbars", and likewise the "A Antiprotons" and "B Protons" boxes were the directions of the diodes. This becomes important when using pickups that result in single-bump signals. This would result in a positive signal for the protons and a negative signal for the antiprotons, which would allow one to use the same set of pickups for both species of particle. With doublets, however, this is somewhat less important and one must only make sure to select the same half of the doublet for subtraction at the front end because the initial and restoring halves of the doublet should be of roughly the same magnitude. It is also

necessary to make sure the time constant, τ in *Equation 25*, is neither too long nor too short.

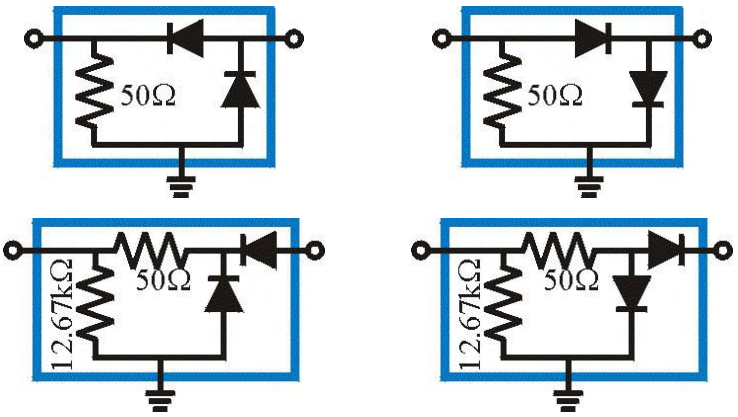

Figure 14 – Diagrams of the four different diode boxes. Clockwise from the upper-left corner these are labeled "A Antiprotons", "B Protons", "Protons", and "Pbars".

Because the diode boxes will work with either species of particle using the given pickups, the species named on the diode boxes is of little consequence. To simplify naming conventions somewhat, the diode boxes with the 50 Ω shunt resistance will be referred to as the Low Shunt Impedance, or LSI, boxes and the diode boxes with the 12.67 kΩ shunt resistor will be referred to as the High Shunt Impedance, or HSI, boxes. It was found that tunes were most easily seen using the LSI boxes so the HSI boxes were not used for any of the tune analysis.

Taking a sample measurement of the signal following the LSI boxes, a 'saw tooth' pattern could be seen as shown in *Figure 15*. As expected from *Equation 26*, the signal jumps to a value, which happens to be negative indicating that either the LSI box labeled "A Antiprotons" or the HIS box labeled "Pbars" were used. The signal then decays until the next bunch passes the detector and the signal reaches a different maximum point before decaying again.

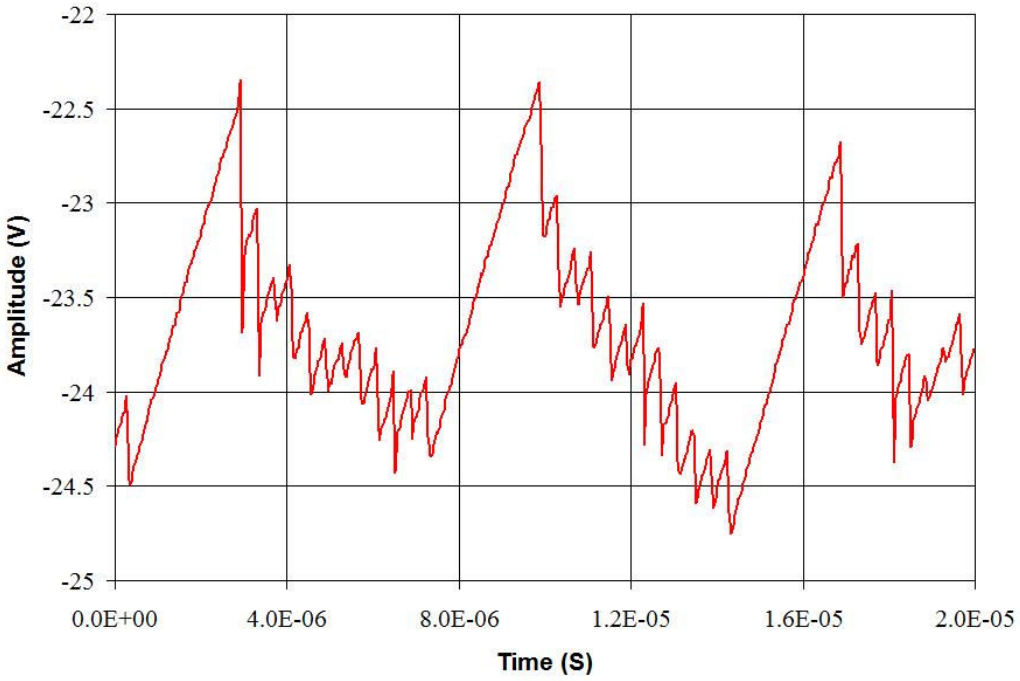

Figure 15 – The proton signal following an LSI diode box. The period shown is 20μs, or almost one revolution period in the Tevatron. The longer periods of potential decay occur between bunch trains as seen in *Figure 1*. This measurement is highly averaged to see the general behavior of the diode box output. Since the triggering on the scope was adjusted to give a preferential position to an edge of the bunch trains these can be seen clearly, but discrepancies result from the samples that did not trigger with the edge of a bunch train as seen as a rather erratic RC decay time constant. A much cleaner signal could be obtained with an external trigger locked to the RF.

In principle one need only adjust the shunt resistance, R_f, inside the diode box as seen in *Figure 16* to tweak the time constant. While decreasing the overall resistance of the network is relatively easily executed, by placing resistors to ground in parallel with R_f, increasing the resistance, and therefore the time constant, requires changing the resistors inside the box. This also places the rest of the diode box circuit at risk and was generally avoided.

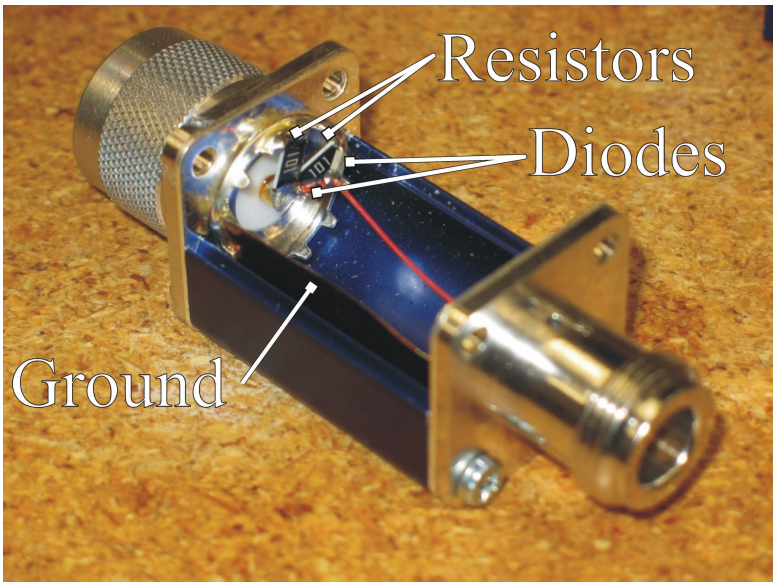

Figure 16 – The inside of one of the LSI Diode Boxes. The shunt resistor network consists of two 100 Ω surface resistors in parallel for a total shunt resistance R_f of 50 Ω

The shunt capacitance, C_f, on the other hand, comes from the length of cable between the diode box and the BBQ front end. The typical capacitance for an RG-58 cable is roughly 30 $^{pF}/_{ft}$ and typical signal propagation speeds are roughly 0.66 $^{ft}/_{ns}$. This yields a shunt capacitance of about 158 pF for an 8 ns cable– the length recommended for this exercise. The resulting time constants, τ, and their corresponding degrees of suppression, ξ, are summarized in *Table 1*. The normalized time constants are shown for the revolution period, $T = 21$ μs, and for the inner-bunch period, $T_{b\text{-}b} = 395$ ns, for the 21 RF buckets between proton or antiproton bunches. In either case, the normalized time constant will be rather small resulting in little suppression of the revolution frequency according to *Equation 36*.

Table 1 – Summary of diode detectors and their respective time constant

Diode Detector	R_f	τ	τ / T	ξ	$\tau / T_{b\text{-}b}$	$\xi_{b\text{-}b}$
HSI Diode Boxes	12.67 kΩ	2 μs	$9.5 \cdot 10^{-2}$	1.12	5.1	20.3
LSI Diode Boxes	50 Ω	7.9 ns	$3.8 \cdot 10^{-4}$	1.00	0.02	1.01

Increasing the shunt capacitance is significantly easier than increasing the shunt resistance, and can be done by increasing the cable length between the diode box and the

front end or adding physical capacitors in parallel with R_f and C_f. Increasing the cable length will eventually become difficult to manage due to the attenuation in the cable, timing differences between the two legs in the circuit, and interference picked up by the unnecessarily long cable. The RG-58 cable used is especially notoriously susceptible to noise when placed in an extended network or loops.[vi] The alternative is to fabricate component boxes with the desired shunt capacitance and to place them in line with the diode detectors. This was done with a shunt capacitance of 4.05μF as shown in *Figure 17.* As expected, this greatly increased the time constant, from 7.9ns to 203μs according to *Equation 25*. This results in a normalized time constant $\tau/T = 9.6$ which brings the suppression factor, ξ, up to about 38 according to *Equation 36*. It also resulted in far more ringing and a low-frequency oscillation, as seen in *Figure 18,* but even if the ringing were not an issue the question remains of what the limits of the time constant might be.

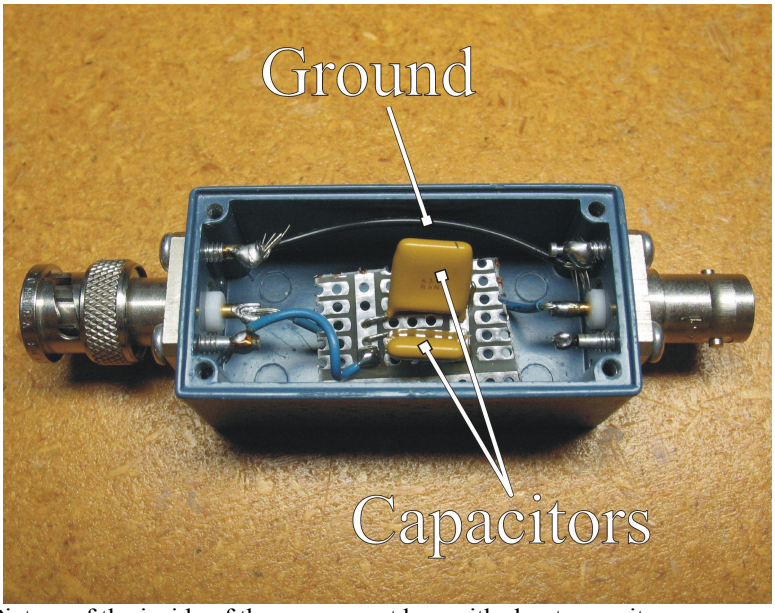

Figure 17 – Picture of the inside of the component box with shunt capacitors.

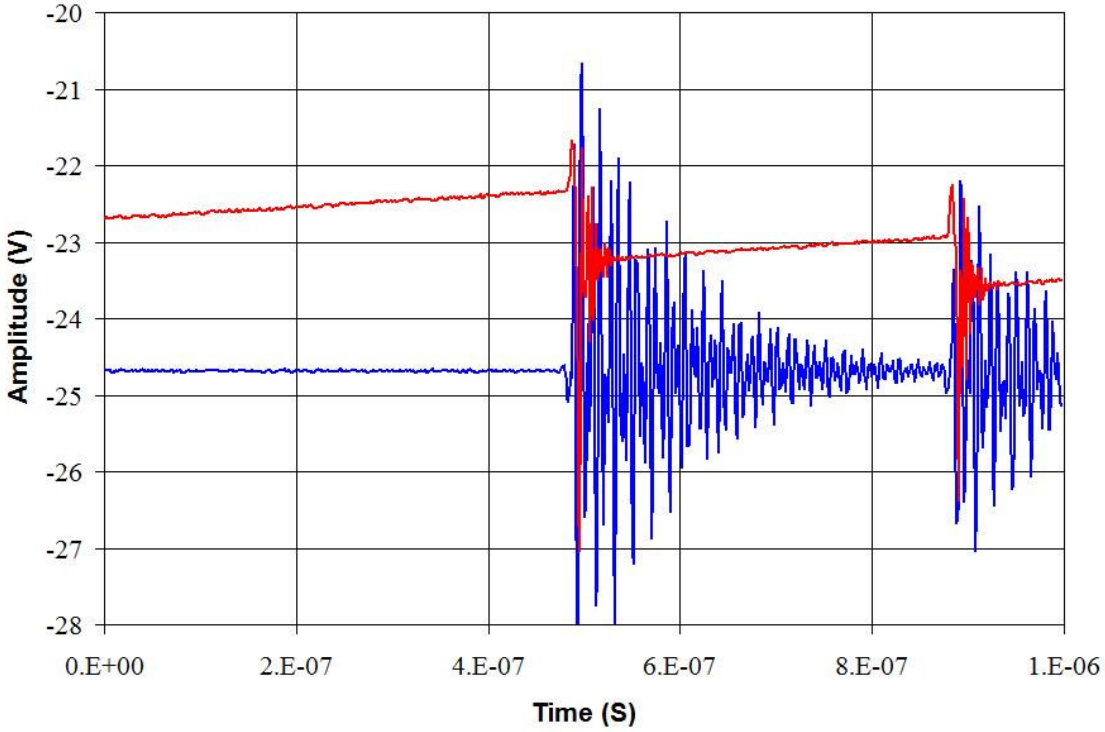

Figure 18 – Increasing the shunt capacitance following the diode detectors results in increasing the decay time of the ringing and results in a smaller-amplitude bunch-to-bunch oscillation. The red trace is with no additional shunt capacitance following the diode detector while the blue trace has an additional 4.05 μF shunt capacitance. The 158 pF shunt capacitance from the standard 8 ns RG-58 cable would fall between the other traces both in terms of ringing amplitude and potential decay rate.

The immediate limiting conditions come from the beam signal itself. If too large of a time constant is used, the signal incident on the shunt capacitor from the pickups will jump to the amplitude for a bunch before the capacitor can sufficiently discharge to even register a voltage change. This will result in missed bunches, referred to as "dragging". While the tune will still appear in the event of dragging, it will not be as prominent on the unclipped signal due to loss of signal information. Too short of a time constant, however, may result in merely following tracking the trailing edge of the bunch signal at which point the function of peak detection is lost. More subtly, the point at which the decaying signal is allowed to reach equilibrium between bunch signals is also important, in which case any suppression of the revolution frequency at the diode detectors is lost. A capacitor is considered to be 'fully' discharged through a resistor after a period of $4 \cdot \tau$.[vii]

To keep the peak signal from decaying to 0 before the next bunch arrives, the time constant must be greater than

$$\tau_{min} = n_{b-b} \Big/ (4 \cdot f_{RF}) \tag{37}$$

where n_{b-b} is the greatest number of RF buckets between bunches. With a single bunch n_{b-b} is 1113, or the harmonic number of the Tevatron. During normal store operations, however, the largest gap is between bunch trains with 140 RF buckets which yields a τ_{min} of 0.66μs. This is much greater than the LSI time constants seen in *Table 1* and yet the inter-bunch plots seen in *Figures 15* and *18* indicate that the time constants are much longer than the calculations from *Equation 25* would indicate. With no explicit shunt capacitance, for the red trace in *Figure 18*, the time constant should be close to zero, so one would expect the trace to follow the negative half of the doublet shown in *Figure 15*. This discontinuity remains somewhat of a mystery.

Other manipulations of the circuit can also be performed to improve the signal. These include the addition of filters and attenuators prior to the diode boxes. The principle of adding a filter before the diode box is primarily an attempt to limit the bandwidth of the signal before the diode, thus improving the signal to noise ratio. Following the peak detector the common mode is no longer necessary and so it may also be filtered, which will improve the front end bandwidth.[viii] Adding attenuators between the pickups and diode boxes will decrease the peak amplitude. With smaller signals incident on the diode detectors, the steps from peak to peak become smaller, which further aid attenuation of the revolution frequency. More importantly, however, this could be useful in the event that the pulse amplitude exceeds the breakdown voltage of the diodes in the diode detector, or in the event that a saturation level is reached in the diode detectors or front end due to the beam signal. Because the signal is carried by the beam, the amplitude of the signal will increase with the number of particles passing the pickup that carry the signal. One could then track saturation of the diode detectors as a function of a ‘bunch current’. The resulting circuit will appear as shown in *Figure 19*.

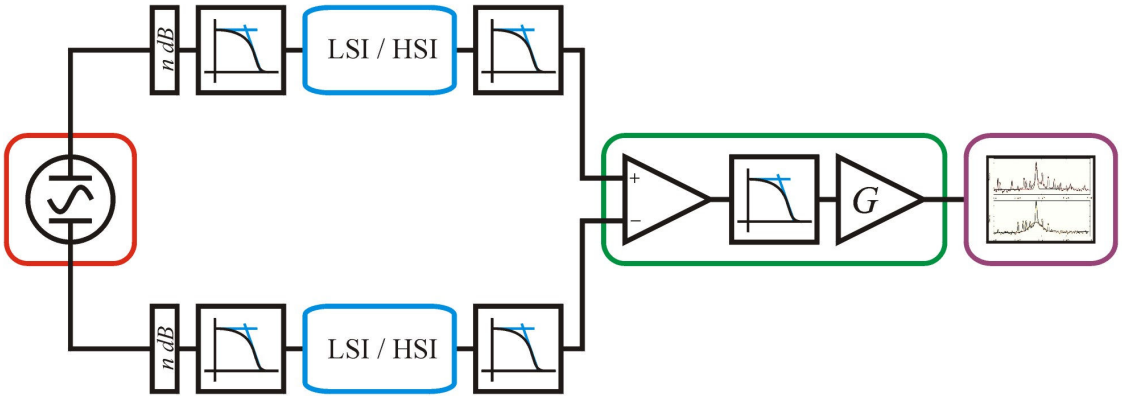

Figure 19 – The BBQ circuit including filters and attenuators. The attenuator is shown as an "*n* dB" pad before any pre-diode box filtering. 70 MHz low pass filters were generally employed prior to the diode boxes to maximize the diode bandwidth. As suggested, there may also be filtering between the diode box and front end to further attenuate the revolution frequency, which were not used.

The 'bunch current' is a term used here to mean the average beam intensity normalized to the average bunch length. Using readbacks from the Tevatron Shot Bunch Display, or SBD, the bunch current can be found for either protons or antiprotons. The ACNET parameters for number of particles are T:SBDPIS and T:SBDAIS for protons and antiprotons respectively, which are read back in units of 10^9 particles. The parameters for bunch length are T:SBDPWS and T:SBDAWS for protons and antiprotons respectively, and read back in units of nanoseconds. Because T:SBDPIS and T:SBDAIS give the total number of their respective particle in the Tevatron, the average will be $^1/_{36}{}^{\text{th}}$ of that and therefore the proton and antiproton 'bunch currents' will be

$$\left.\begin{aligned} J_B^+ &= {}^{T:SBDPIS}\!/_{(T:SBDPWS\cdot 36)} \\ J_B^- &= {}^{T:SBDAIS}\!/_{(T:SBDAWS\cdot 36)} \end{aligned}\right\} \qquad (38)$$

respectively. Because of the scaling of the SBD parameters, the bunch current will be in units of 10^{18} particles per second, or Exahertz (EHz).

Early measurements suggested that with no attenuation or filtering the BBQ system would saturate with a proton bunch current around an average of 93 EHz.[ix] If the bunch current is higher than this, the tune will be buried in the noise floor, inhibiting measurement of the tune. As the store progresses protons are naturally lost, which lowers the average number of protons per bunch, $^{T:SBDPIS}/_{36}$, and will eventually the bunch current will fall below its upper threshold allowing measurement of the tunes. Alternately, the signal can be attenuated, effectively simulating a smaller number of protons per bunch. The desired attenuation factor, A, is simply the ratio of the maximum initial store bunch current to the threshold bunch current,

$$A = {J_{\max}}\Big/{J_{thr}} \tag{39}$$

For typical Run IIB average proton bunch intensities and lengths of $300 \cdot 10^9$ and 1.7 *ns* respectively the maximum initial store bunch current will be on the order of 176 EHz. Given the measured threshold bunch current, J_{thr}, of 93 EHz, the signal requires attenuation by at least a factor of 1.90. This corresponds to $10 \cdot Log_{10}(1.9) \approx 2.8$ dB of attenuation. Given this, a 3dB pad should be sufficient to ensure that the proton signal will not exceed the maximum bunch current threshold.

One should note that this will change for different diode detectors as well as between planes. For example, when using no attenuation before the diode boxes the horizontal tunes were visible well before the vertical tunes. This suggested that the upper bunch current threshold for the vertical tune appeared to be lower than that of the horizontal, which required additional attenuation. Therefore 3 dB was added to each of the horizontal legs and 6 dB was added to each of the vertical legs before the diode boxes for initial data collection to ensure that the upper threshold bunch current was not approached in either plane.

The concept behind the BBQ front end is comparatively simple. The difference of the two signals is taken, filtered, amplified and passed on for Fourier analysis. Like the diode boxes, the front end also suppresses the revolution frequency. A frequency

response on one of the BBQ front end channels, shown in *Figure 20*, indicates that the revolution frequency should be suppressed by about 90 dB on the output of the BBQ front end. In spite of this it may still suffer from saturation like the diode boxes so the gain levels were each checked to find where saturation may become an issue.

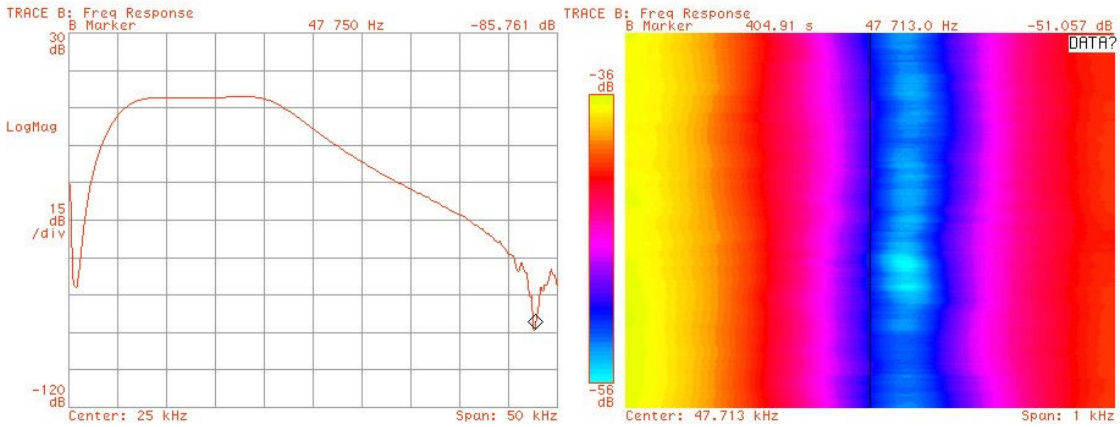

Figure 20 – To the left is a typical frequency response of the BBQ front end with a random noise input from an HP89410A VSA. The tunes in the Tevatron are generally within a couple of kHz of 19.5 kHz, near the edge of the plateau. The revolution frequency, 47.713 kHz, is about 90 dB below the betatron tunes. This attenuation will change to some extent since the band of highest attenuation is slightly higher than 47.713 kHz as shown in the spectrograph to the right.

The four gain levels, labeled 0 to 3 for each plane, were checked to ensure that there was no saturation at the BBQ front end. An input peak voltage was measured from the pickups, V_1, and then compared against the output voltage, V_2. Once each of the amplification levels was measured, summarized in *Tables 2* and *3*, the combinations of amplifications were compared to the sums of the individual amplifications. There may have been some saturation at the higher amplification settings as indicated by a greater deviation between the sum of the individual gains and the measured combination but this may have also been due to the scope being used. In practice the final amplification settings were chosen such that the input to the VSA was at about half of its saturation power with input from the BBQ front end and this rarely required over 30dB of gain, which is well under any saturation level.

Table 2 – Horizontal Channel Gains

Dip Switch Configuration	V_2 [µV]	V_2/V_1	Measured Gain [dB]	Calculated Gain [dB]	Variation [%]
None	51	1	0		
0	90	1.76	4.91		
1	170	3.33	10.45		
2	370	7.25	17.21		
3	1700	33.33	30.46		
0+1	290	5.69	15.1	15.36	1.7
0+2	600	11.76	21.41	22.12	3.3
0+1+2	2000	39.22	31.87	32.57	2.2
1+2	1200	23.53	27.43	27.66	0.8
0+3	3100	60.78	35.68	35.37	0.9
1+3	5700	111.76	40.97	40.91	0.1
2+3	10700	209.8	46.44	47.66	2.6
0+2+3	15500	303.92	49.66	52.57	5.9
1+2+3	16900	331.37	50.41	58.11	15.3
0+1+2+3	16800	329.41	50.35	63.02	25.2

Table 3 – Vertical Channel Gains Measured on the VSA

Dip Switch Configuration	V_2 [µV]	V_2/V_1	Measured Gain [dB]	Calculated Gain [dB]	Variation [%]
None	5.4	1	0		
0	8.6	1.59	4.04		
1	15.6	2.89	9.21		
2	27.3	5.06	14.08		
3	149	27.59	28.82		
0+1	27	5	13.98	13.26	5.2
0+2	45.5	8.43	18.51	18.12	2.1
0+1+2	152	28.15	28.99	27.33	5.7
1+2	90.1	16.69	24.45	23.29	4.7
0+3	270	50	33.98	32.86	3.3
1+3	295	54.63	34.75	38.03	9.4
2+3	784	145.19	43.24	42.89	0.8
0+2+3	1410	261.11	48.34	46.93	2.9
1+2+3	1761	326.11	50.27	52.11	3.7
0+1+2+3	2090	387.04	51.76	56.15	8.5

In the time domain, the front end output signal looks like noise, being a convolution of all the signals, betatron oscillation, chromaticity, and other beam properties along with signals from in-line and surrounding electronics. The BBQ design report steps through the process in PSpice with monotonic tunes to reveal that what is left after the front end is approximately a monotonic sine wave of the tune frequency. With all of the convoluted, multi-tonic signals mentioned above, however, picking out the tune in the time domain would be virtually impossible. Instead, it is much more convenient to observe the signal in the frequency domain as seen in the next section.

60Hz Spike Analysis

The most striking feature of the VSA spectrum, shown in *Figure 21*, was the sharp spikes littering the area in which the tunes reside. Similar spikes are seen at Brookhaven at 60Hz intervals, and CERN at 50Hz indicating that they are due to AC line frequency, which runs at 60Hz in the United States and at 50Hz in France and Switzerland. This section is not intended to be a rigorous analysis of the 60Hz, but merely a demonstration of their presence and that such a signal could be placed on the beam. The source of the spikes is hotly debated and beyond the scope of this paper. Initially, it was believed that the spikes were due to the BBQ hardware picking up the AC signal before or even due to the front end. More observations and recent studies have shown that the 60Hz lines appear coupled to the beam. [such as P. Cameron et al, *Observations of Direct Excitation of the Betatron Spectrum by Mains Harmonics in RHIC*]

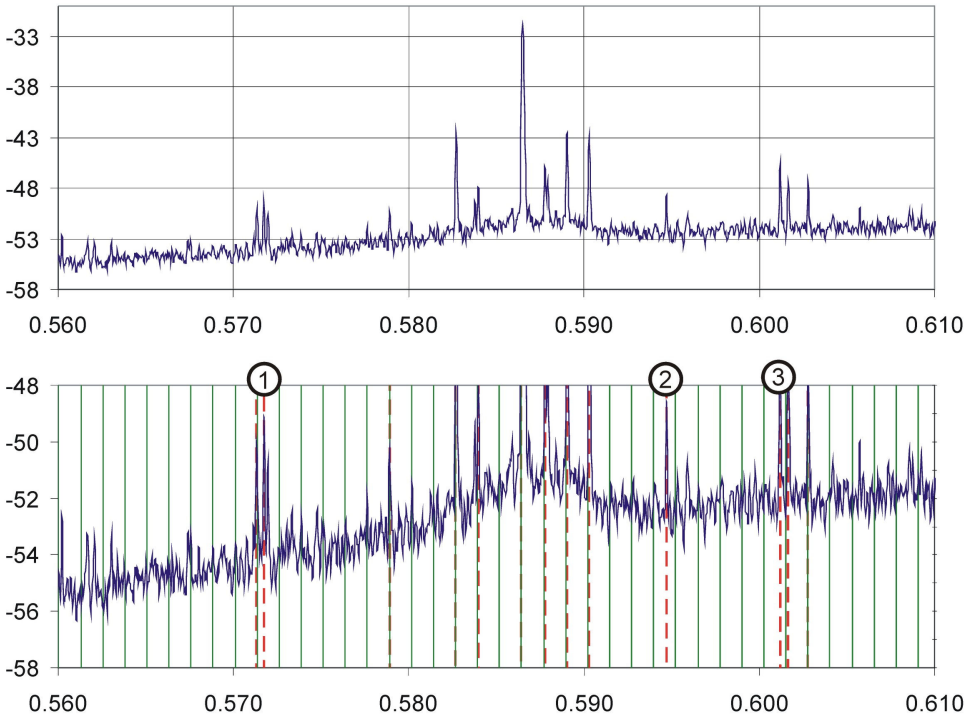

Figure 21 – The top is a spectrum seen during Store 5590 from the Proton Horizontal Pickups through the BBQ front end. Below is the same data. Spikes were selected (denoted by the red hashed lines) and intervals of 60Hz were drawn out from the central, and tallest, spike. Three of

the spikes shown here did not coincide with 60Hz lines. These spikes were at tune/frequency values of ① 0.5717 / 20.433kHz, ② 0.6012 / 19.028kHz, ③ 0.6017 / 19.005kHz respectively.

One such study at Brookhaven National Laboratory (BNL) involved adjusting a single skew quadrupole, not to decouple the beam, but rather to excite a coupled resonance. Like regular quadrupoles, a skew quadrupole is used to adjust the tune of the beam in a synchrotron, but it is rotated by 45 degrees. As such, skew quadrupoles are used to adjust coupling of the tunes, or the interaction of the tunes between the horizontal and vertical planes. Adjusting the skew quad, thereby changing the coupling, resulted in a corresponding change in the amplitude of the 60Hz spikes in the vertical plane, indicating that the 60Hz is carried by the beam.[x]

There is also some movement of the 60Hz peaks, as seen in *Figure 22*. The tallest peak was tracked through Store 5590, and shows some movement over the duration of the store. The movement, however, is not large and does not appear to correspond well to tune changes. For the majority of the store the maximum peak resides in a 0.0007 band around the tune 0.5888. Given a span of 5kHz, this translates to a 10-point change across the 1600 points used in the VSA measurement, or about 31Hz. The tune of 0.5888 corresponds to the 327^{th} harmonic of the base 60Hz frequency, where the same band is only about 0.09Hz wide suggesting that the movements seen corresponds to the drift in the AC line frequency over the period of the store.

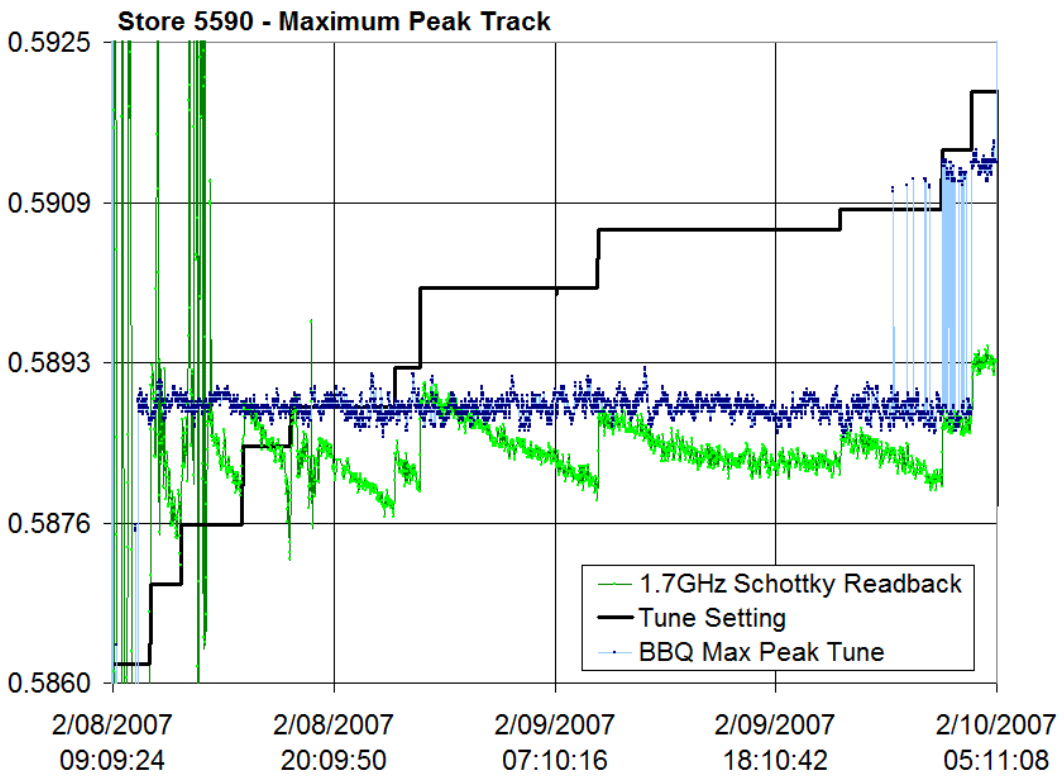

Figure 22 – Tracking the maximum point on the horizontal BBQ signal through data collected during store 5590 as compared to the 1.7GHz Schottky readback and horizontal tune settings.

Since the source of the spikes appears to be on the beam itself, the 60Hz must originate elsewhere and be placed on the beam. A prototypical scenario that would result in the 60Hz spikes on the beam is to consider a source that kicks the beam once every $T_{60} = \frac{1}{60}^{\text{th}}$ of a second. Consider the idealized, single-bunch example shown in *Figure 5*. If the orbit of the bunch were to spontaneously change for a single revolution period, T, the change in the orbit at the pickup would result in a change in the initial voltage at the diode detector in *Equation 30*. Repeating this every T_{60} seconds, as shown in *Figure 23*, will result in a series of kicks to the beam such that the resulting contribution to the diode detector input could be expressed as

$$s(t)_{60} = \left(\sum_{k=-\infty}^{\infty} \left[H\left(t - kT_{60} + \frac{T}{2} \right) - H\left(t - kT_{60} - \frac{T}{2} \right) \right] \right) * \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{40}$$

While T is generally not an integer multiple of T_{60}, assume for a moment that it is. This means that only one out of every T/T_{60} revolutions will be kicked resulting in a series of delta functions, $\sum_{k=-\infty}^{\infty}\delta(t-kT_{60})$, which will be added on to the diode detector input as shown in *Figure 24*. As a result the Fourier transform will result in 60Hz harmonics and *Equation 30* can be rewritten as

$$\left.\begin{aligned} s_d(t) = A_b\cdot\cos(2\pi f_b t)\left[s_f(t)*\sum_n\delta(t-nT)\right] \\ +A_o\left[s_f(t)*\sum_n\delta(t-T)\right] \\ +A_{60}\left[s_f(t)*\sum_k\delta(t-kT_{60})\right] \end{aligned}\right\} \quad (41)$$

where A_{60} is the amplitude of the 60Hz kick to the single pulse.

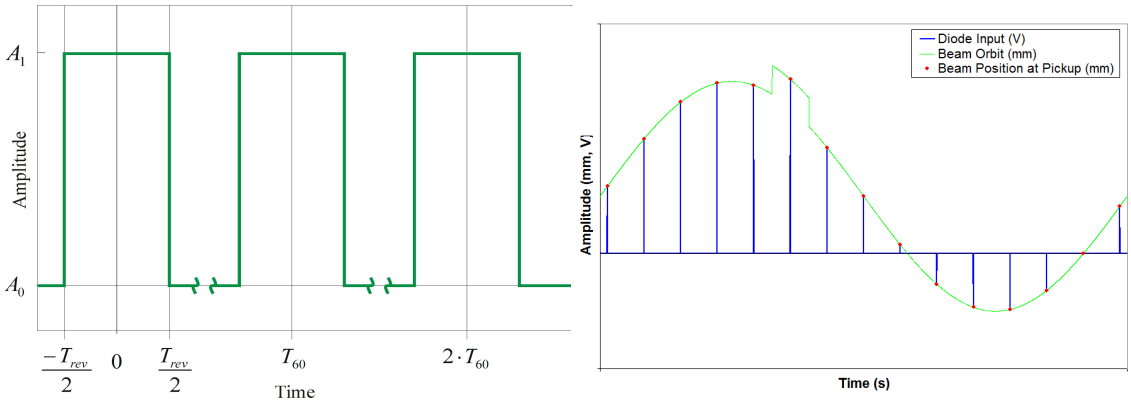

Figure 23 – A particle orbit amplitude that would result in 60Hz harmonics on the Fourier Transform. The heaviside pair has a width of T and periodicity of T_{60}. The result is a change in the orbit every T_{60}.

When a Fourier transform is applied to *Equation 41* the filter hold function envelope ensures that the 60Hz spikes will appear in sideband and common mode spectra. This is only one scenario that will result in 60Hz spikes in the tune spectrum, and there has been extensive speculation and simulations in search of the cause of such a phenomenon, including the interaction of phasing between silicon-controlled rectifiers, or

SCRs, in the mains supplies, which provide current to the main bending supplies. All of the main bend supplies are horizontal, which is what prompted to investigation of the coupling of the 60Hz spikes to the vertical plane previously described. There is some evidence that this is at least a contributing factor, since a minimum in the simulated 60Hz harmonic spectrum can be changed by adjusting the SCR overlap.

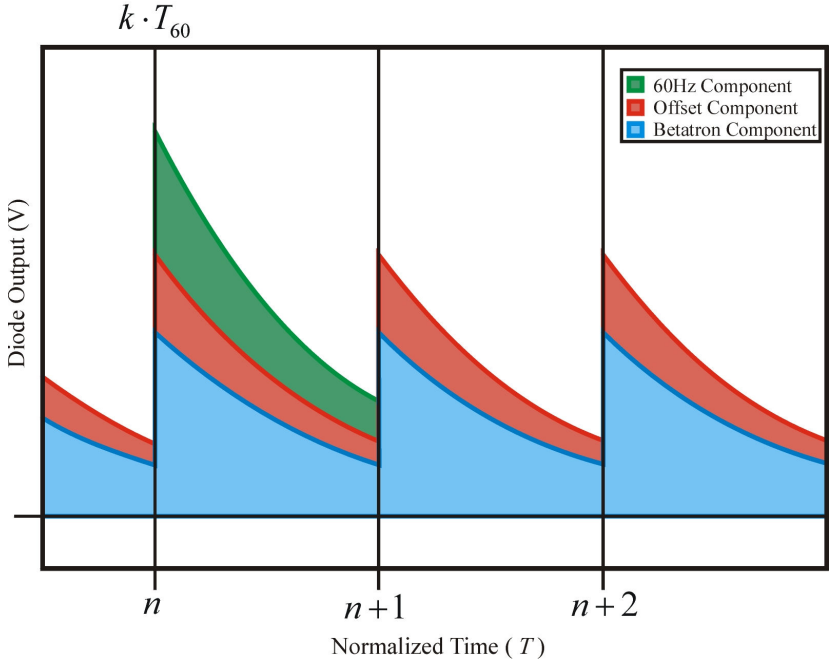

Figure 24 – The diode output broken up into the three components in *Equation 41*. The total output of the diode box is the envelope of the three components.

In practice, this is complicated by the number of magnets applying the 60Hz kick (thousands of Tevatron main bend magnets), bunch distributions, and beating between T and T_{60}. The analysis above also ignores other potential sources of 60Hz noise including 216 dipole correctors, 31 feed-down sextupoles, other correction elements, and powered diagnostics around the Tevatron. It is conceivable that rectification in each of the supplies of the powered elements could kick the beam in a similar manner. Even with all the sources, however, the result is still limited to 60Hz harmonics.

The Levenberg-Marquardt Algorithm

While the spikes do not appear to significantly degrade beam quality, they make measuring the tunes more challenging. Measurement of the tunes in the Recycler Ring at Fermilab, in contrast, is rather trivial. The Schottky signal is taken from a dedicated VSA, RRMCRVSA, and the highest point of an averaged signal is taken to be the tune. There's some sampling noise and tune shift due to the Main Injector ramp, so the average tunes are also made available by means of a weighted average. The spikes on the TeV BBQ data render this approach unusable since the spike closest to the tune peak may or may not have the highest amplitude and, even if it does, it will not track with the tune as demonstrated in *Figure 22*. While the spikes are very tall, they are relatively infrequent making the bulk of the data generally usable if the spikes could be ignored.

The premise of this is fairly straight-forward: find or derive a formula that reliably describes the shape of the data, and then fit the data to that formula. From a purely qualitative perspective, the shape of the tune bump appeared to be approximately Gaussian. Also, while the spectrum in *Figure 21* is clearly not flat, the local baseline could be approximated as linear. The result was the linear combination of a Gaussian with a line:

$$f(x) = p_1 \cdot e^{-\frac{1}{2}[(x-p_2)/p_3]^2} + (p_4 \cdot x) + p_5 \tag{42}$$

where p_n are the independent parameters to be found. There are ways of approximating the linear portion of the fit, especially if the frequency range is wide enough to assume that the entire Gaussian lies within the data range. It would be better, however, to return all five parameters from a fitting routine.

Fitting to five parameters in a non-linear function like the one in *Equation 42* is not trivial and generally requires something more robust than a simple nonlinear least-squares fit. The Levenberg-Marquardt algorithm, or LMA, is a combination of the Gauss-Newton algorithm, or GNA, and gradient descent algorithm that is widely used to fit to

non-linear equations with multiple fit parameters. LMA is an iterative process that favors the gradient descent approach when a local minimum cannot be found but, as a final solution is approached, the GNA gains precedence.[xi]

The ultimate goal of least-squares and related fit algorithms is to minimize the sum of the squares of the deviations between the fit, $f(x_i, \vec{p})$, and the data, y_i, where $i = (1,\ldots,n)$, n being the number of data points to fit to.

$$S(\vec{p}) = \sum_{i=1}^{n} [y_i - f(x_i, \vec{p})]^2 \tag{43}$$

where the fit parameters are the set $\vec{p} = (p_1,\ldots,p_m)$. If the parameters, $\vec{p}$, are perturbed by $\vec{\delta} = (\delta_1,\ldots,\delta_m)$,

$$f(x_i, \vec{p} + \vec{\delta}) \approx f(x_i, \vec{p}) + J_i\vec{\delta} \tag{44}$$

to the first order by Taylor Expansion where the Jacobian, J, is

$$J = \begin{pmatrix} \frac{\partial f(x_1)}{\partial \vec{p}} \\ \vdots \\ \frac{\partial f(x_n)}{\partial \vec{p}} \end{pmatrix} = \begin{pmatrix} \frac{\partial f(x_1)}{\partial p_1} & \cdots & \frac{\partial f(x_n)}{\partial p_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(x_1)}{\partial p_m} & \cdots & \frac{\partial f(x_n)}{\partial p_m} \end{pmatrix} \tag{45}$$

For example, the partial differentials of *Equation 42* with respect to the fit parameters used to fill out the Jacobian are

$$\left.\begin{aligned} \frac{\partial f}{\partial p_1} &= e^{-z^2/2} \\ \frac{\partial f}{\partial p_2} &= p_1 e^{-z^2/2}\left(\frac{x-p_2}{p_3^2}\right) \\ \frac{\partial f}{\partial p_3} &= p_1 e^{-z^2/2}\left[\frac{(x-p_2)^2}{p_3^3}\right] \\ \frac{\partial f}{\partial p_4} &= p_4 \\ \frac{\partial f}{\partial p_5} &= 1 \end{aligned}\right\} \tag{46}$$

where, $z(p_1, p_2) = \left(\frac{x-p_1}{p_2}\right)^2$.

If the perturbation $\vec{\delta}$ was just right, it would minimize *Equation 43* such that:

$$\left.\begin{aligned}\frac{\partial S}{\partial \delta} &= \frac{\partial}{\partial \delta}\sum_i \left[y_i - \left(f\left(x_i, \vec{p}+\vec{\delta}\right)\right)\right]^2 \\ &= \frac{\partial}{\partial \delta}\sum_i \left[y_i - \left(f\left(x_i, \vec{p}\right)+J_i\vec{\delta}\right)\right]^2 \\ &= 2\cdot\sum_i \left[J_i\left(\left[y_i - f\left(x_i, \vec{p}\right)\right]-J_i\vec{\delta}\right)\right] = 0\end{aligned}\right\} \quad (47)$$

Eliminating the constant and considering the vector form, *Equation 47* can be rewritten as

$$J \circ \left(\left[\vec{y} - f\left(\vec{x}, \vec{p}\right)\right]-\left(J\vec{\delta}\right)\right) = 0 \quad (48)$$

But $a \circ b = a^T b$ where a & b are column vectors and $a(b+c) = ab + ac$ for matrices a, b and c. Using these identities and rearranging *Equation 48*,

$$J^T\left[\vec{y} - f\left(\vec{x}, \vec{p}\right)\right] = \left(J^T J\right)\vec{\delta} \quad (49)$$

which is the Gauss-Newton Algorithm. While, strictly speaking, the Jacobian, *J*, is an *n* by *m* matrix, it can be written as a 1 by *n* column vector as well when considering only the J_i as has been done above.

The GNA can be difficult to use, particularly when starting with initial fit parameters that may be very far from their best fit values. Seeing this, Levenberg added a damping factor, λI_m, where λ is a scalar that can be adjusted from iteration to iteration and I_m is an *m* by *m* identity matrix. Rearranging and adding in the damping factor, *Equation 49*, becomes

$$\left(J^T J + \lambda I_m\right)\vec{\delta} = J^T\left[\vec{y} - f\left(\vec{x}, \vec{p}\right)\right] \quad (50)$$

For large λ, the component $J^T J$ disappears and the following expression remains

$$I_m \vec{\delta} = \tfrac{1}{\lambda}\left(J^T\left[\vec{y} - f(\vec{x}, \vec{p})\right]\right) \tag{51}$$

This has similar formulation to the gradient descent method which states that a minimum can be iteratively found if one follows the negative gradient to a minimum. One can verify *Equation 51* by taking the gradient of *Equation 43* with respect to the fit parameters, $\vec{p}$, such that

$$\frac{\partial S}{\partial \vec{p}} = \frac{\partial}{\partial \vec{p}} \sum_i \left[y_i - f(x_i, \vec{p})\right]^2 = -2 \cdot \sum_i \left(\frac{\partial f}{\partial \vec{p}} \left[y_i - f(x_i, \vec{p})\right] \right) \tag{52}$$

But $\frac{\partial f(x_i, \vec{p})}{\partial \vec{p}} = J_i$ so moving into vector formulation it can be seen that,

$$\frac{\partial S}{\partial \vec{p}} = -2\left(J \circ \left[\vec{y} - f(\vec{x}, \vec{p})\right]\right) = -2\left(J^T\left[\vec{y} - f(\vec{x}, \vec{p})\right]\right) \tag{53}$$

The negative gradient, required for gradient descent, is

$$-\frac{\partial S}{\partial \vec{p}} = 2\left(J^T\left[\vec{y} - f(\vec{x}, p)\right]\right) \tag{54}$$

For $^1/_\lambda = 2$, *Equation 54* is identical formulation to *Equation 51*, which is therefore also a gradient descent.

Marquardt noted that this formulation does not scale for vast differences in parameter gradients. It was suggested that the individual parameter gradients could be taken into account by using the diagonal components of the Hessian, H^d:

$$H = J^T J = \begin{pmatrix} \frac{\partial^2 f}{\partial p_1^2} & \cdots & \frac{\partial^2 f}{\partial p_1 \partial p_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial p_m \partial p_1} & \cdots & \frac{\partial^2 f}{\partial p_m^2} \end{pmatrix} \rightarrow H^d = \begin{pmatrix} \frac{\partial^2 f}{\partial p_1^2} & 0 & \cdots & 0 \\ 0 & \frac{\partial^2 f}{\partial p_2^2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \frac{\partial^2 f}{\partial p_m^2} \end{pmatrix} \tag{55}$$

which leads to the iterative condition for the Levenberg-Marquardt Algorithm,

$$\left(H + \lambda H^d\right)\vec{\delta} = J^T\left[\vec{y} - f\left(\vec{x}, \vec{p}\right)\right] \tag{56}$$

The Hessian, *H*, should not to be confused with the Heaviside step function, $H(t)$, from *Equation 6*. To iterate through the fitting algorithm, initial fit parameters and an initial λ are applied and an appropriate perturbation, $\vec{\delta}$, is found. For subsequent iterations λ is adjusted appropriately and the parameters are adjusted such that

$$\vec{p}_{next} = \vec{p} + \vec{\delta} = \begin{pmatrix} p_1 + \delta_1 \\ \vdots \\ p_m + \delta_m \end{pmatrix} \tag{57}$$

If the new fit parameters result in too small of a change, as compared to a minimum tolerance, λ may be further adjusted. The algorithm is generally terminated once one of the following conditions is met:

- The difference of the squares, $S(\vec{p})$ from *Equation 43*, is sufficiently small
- The change in $S(\vec{p})$ is below a minimum tolerance
- The algorithm has looped a maximum number of times without convergence

The limits for each of these conditions are decided upon prior to entry into the fitting algorithm and are generally based on the precision required, the amount of time available to provide fit parameters, and other factors. Once one of these conditions has been met, the current iteration is typically aborted and the fit parameters for the previous iteration are accepted as the final fit parameters. There are several mathematical packages

available, such as GNUPlot, ROOT, and Matlab that are able to fit to a non-linear expression like the one in *Equation 42*. Root is even portable, but requires proficiency in C++ as well as a detailed knowledge of its own structure to apply it easily. There are also specifically built C++ treatments available and one of these[xii] was selected for use in the stand-alone analysis programs and the eventual development of tune analysis software for the Fermilab accelerator controls system, ACNET.

Analysis using LMA

The Levenberg-Marquardt Algorithm takes all points into account, so the spikes seen in the data in *Figure 21* would tend to result in a fit that accounts for both the spikes and the intended tune data, as shown in *Figure 25*. To analyze the effects of spikes on the fit and further characterize LMA, three data sets were fabricated consisting of the linear combination of a well-defined, relatively broad Gaussian, representing the tune Gaussian seen on the VSA, and a sharp, Gaussian spike, as shown in *Figure 26*. The tune Gaussian was kept fixed through all three datasets. Fitting to this curve alone returned the appropriate values as expected. The amplitude, tune, and standard deviation, p_1, p_2 and p_3 respectively, of the spike Gaussian were each adjusted in turn as shown in *Figures 27*, *28* and *29*. Since the primary concern was with the fit to the Gaussians, both the linear slope and linear offset components in the tune Gaussian, p_4 and p_5 from *Equation 42*, were allowed to be zero. The same initial fit parameters were used at each step for all three datasets. These and the default Tune Gaussian and Spike Gaussian parameters are listed in *Table 4*.

Table 4 – Default parameters for data sets shown in *Figures 27*, *28* and *29*

		Tune Gaussian	Spike Gaussian	Initial LMA Guess
Gaussian Amplitude	p_1	4	40	5
Tune	p_2	50	55	51
Standard Deviation (σ)	p_3	5.000	0.005	1.000
Linear Slope	p_4	0	0	0
Linear Offset	p_5	0	0	0

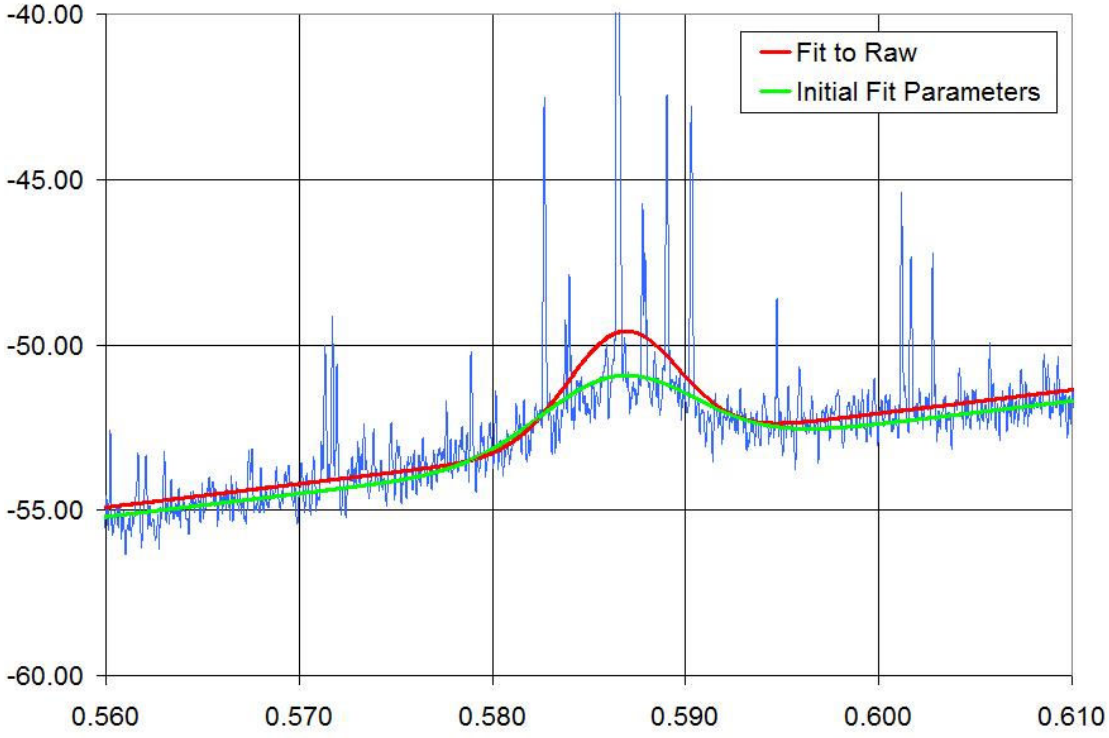

Figure 25 – The same data used in *Figure 21* was fit to by hand, shown as the green trace. Using the resulting fit parameters the data was then fit to using LMA. The resulting fit, the red trace, show a larger Gaussian amplitude due to the spikes.

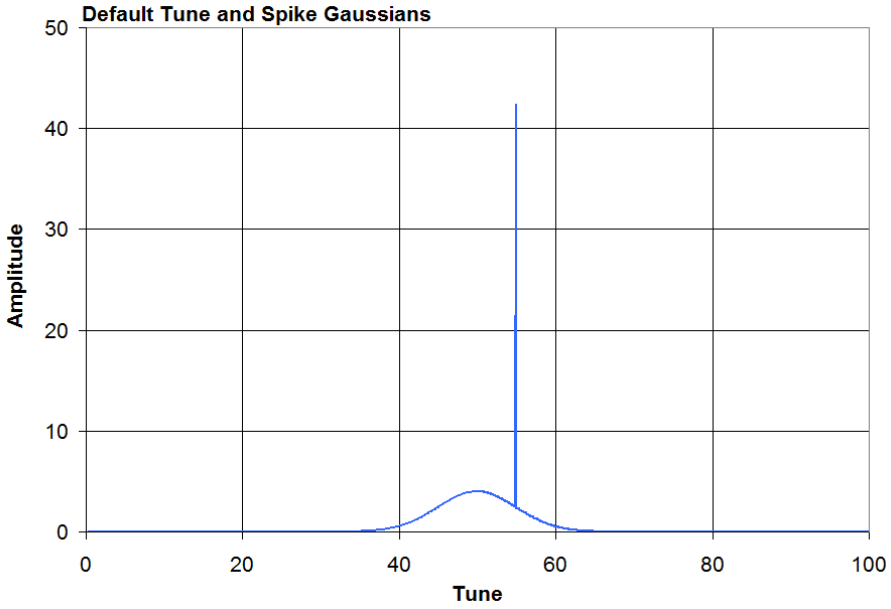

Figure 26 – The default tune and spike Gaussians as specified in *Table 4*.

In the first set, the center and standard deviation of the spike Gaussian were held at their default values, while the amplitude of the spike was increased from 0 to 200 units in 100 steps. The resulting fit parameters start at the initial values of the tune Gaussian as expected and then as they are affected by the growing spike Gaussian, as seen in *Figure 27*. As the spike Gaussian continues to grow it gains precedence and eventually takes over as the dominant spike, even though most of the data is on the tune Gaussian. At that point the amplitude and linear parameters become noisy due to the disparity between initial and final fit parameters as well as terminal fit conditions being met before a more exact fit can be made.

In the second set, the amplitude and standard deviation of the spike Gaussian were held at their default values, while its center was moved across the Gaussian in 100 steps. The amplitude of the spike Gaussian was not large enough to dominate the fit through the set, but evidence of the movement across the tune span was apparent as seen in *Figure 28*. The amplitude reached a maximum as the spike aligned with the center of the tune Gaussian. At the same time, the standard deviation, σ, reached a minimum to correspond with the small standard deviation of the spike Gaussian and the tune for both the tune Gaussian and the spike Gaussian are at 50 units. The linear component fits were asymmetric due to the offset of the initial guess from the center of the tune Gaussian.

In the third set, the amplitude and center of the spike Gaussian were held at their default values and the standard deviation of the spike Gaussian was increased from 0 to 0.5 units in 100 steps. Much like in the first set, the increase in the standard deviation of the spike Gaussian caused the LMA to fit to it rather than the tune Gaussian. It should be noted that the standard deviation of the spike Gaussian in the set summary, in *Figure 29*, is exaggerated to show the change from step to step.

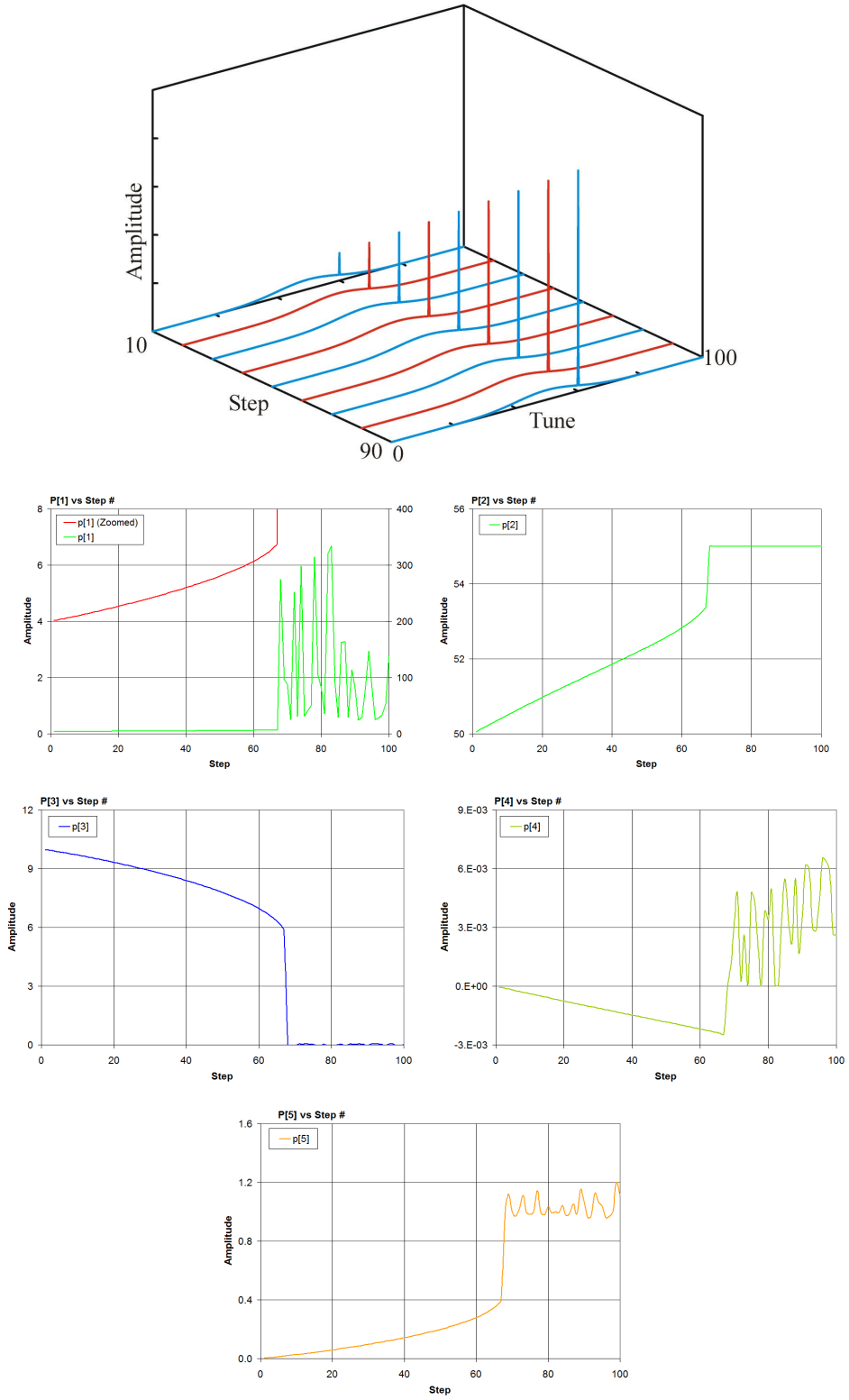

Figure 27 – The graphs of the fit parameters above show that as the spike Gaussian amplitude increases, the LMA adjusts to take the spike into account until it dominates the fit starting at around step 65.

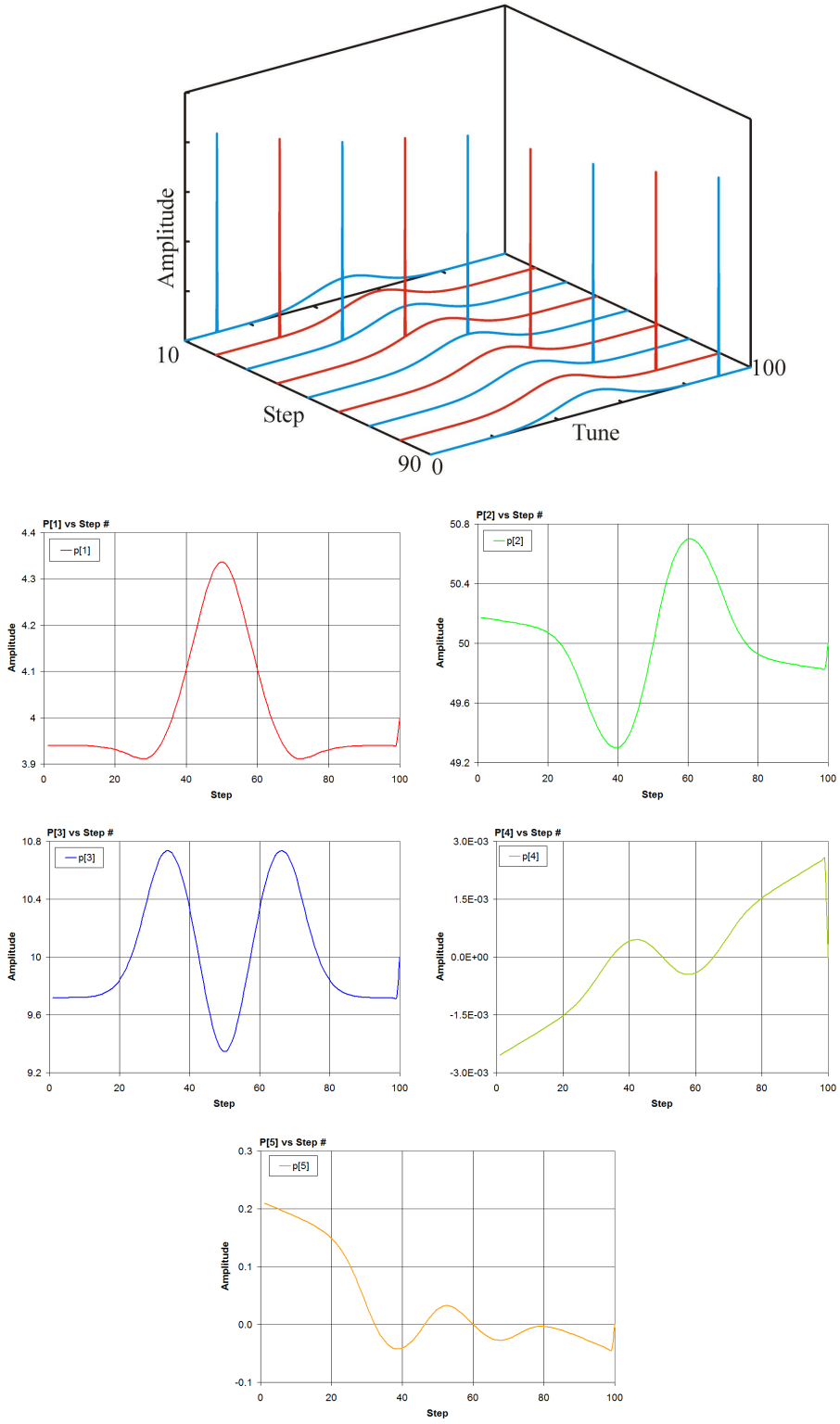

Figure 28 – The graphs of the fit parameters above show that as the spike Gaussian moves across the tune range the fit parameters track its progress. The asymmetry in p_4 and p_5 are due to the initial guess for p_2 being slightly high as detailed in *Table 4.*

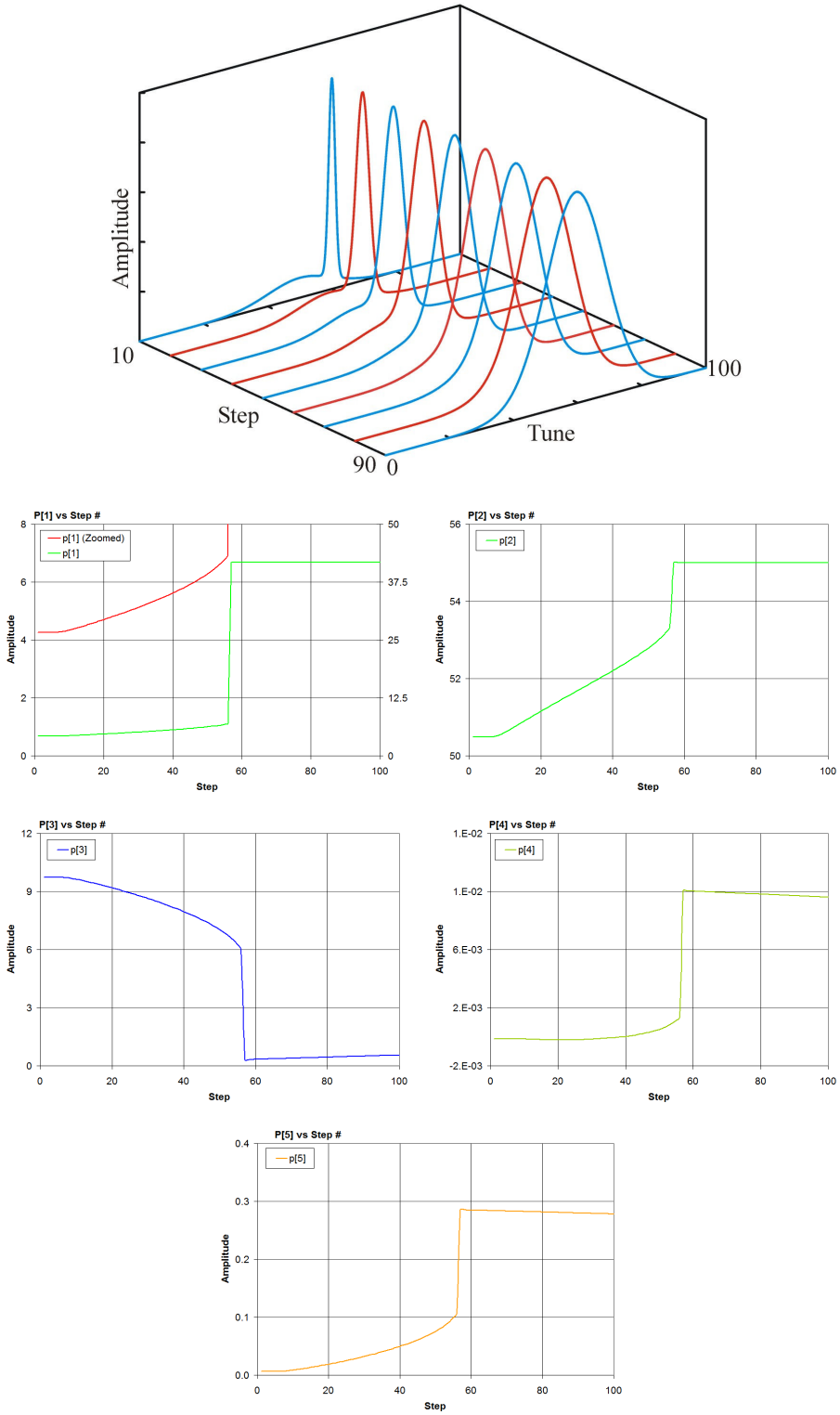

Figure 29 – The graphs of the fit parameters above show that as the σ of the spike Gaussian expands the LMA adjusts to take the spike into account until it dominates the fit starting at around step 55. This is similar to the LMA fits in *Figure 27*.

Data Collection

The examples in *Figures 27, 28,* and *29* confirmed that the spikes seen in *Figure 21* would bias the resulting fit parameters and demonstrated the need for a means of removing them. To this end three methods were developed and evaluated using traces collected from Store 5590 from 08:12:00 CST on 2/8/2007 to 09:09:57 CST on 4/8/2007. The traces were saved as tune-amplitude pairs for analysis using LMA.

First, the raw data including the spikes, were fit to for later comparison with the three methods of spike removal. The fits to the raw data deviated a bit from the readback from the 1.7GHz Schottky, as shown in *Figure 30*, but they appeared to jump appropriately with tune changes to some extent as well, as one would expect. In spite of these features, the fits were not representative of the data, with the amplitude and tune of the fit Gaussian offset in both planes by the spikes. As a result the fit amplitude was made artificially large, and offset tune compared to the 1.7GHz Schottky readbacks.

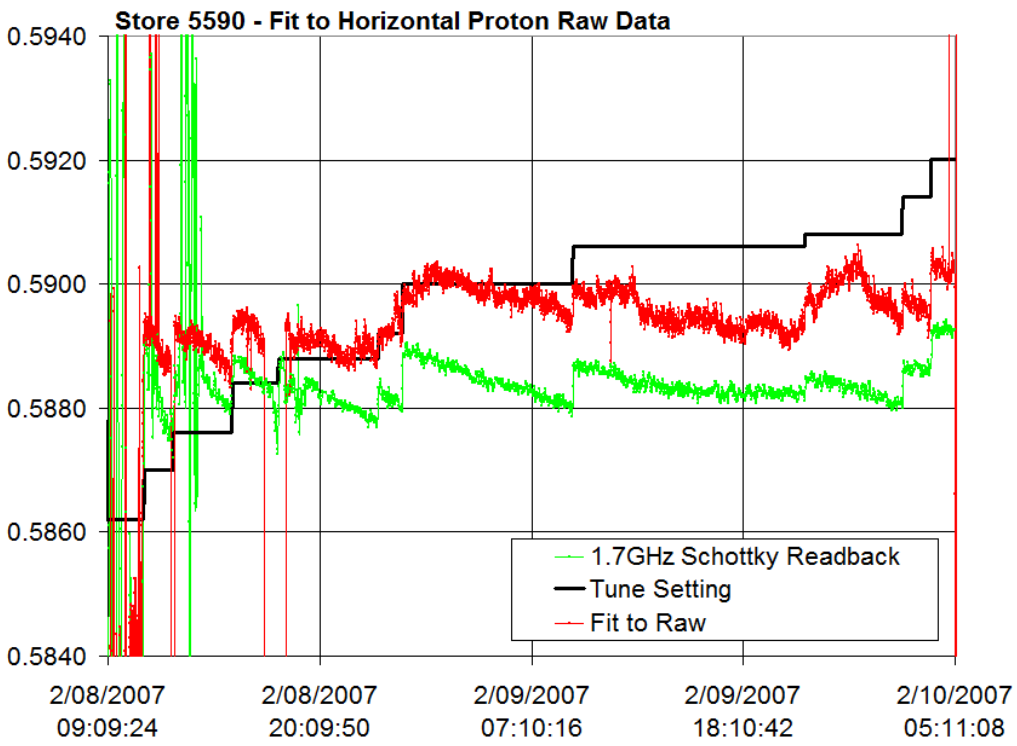

Figure 30 - The tune parameter from the LMA fit to the raw data (Red). The datalogged tune from the 1.7GHz Schottky and the datalogged tune settings are also shown (Green).

The three methods of spike-stripping investigated are referred to as 'Linear-Replacement', 'Omission', and 'Fit-Threshold' spike stripping. The Linear-Replacement method begins by first finding a spike, which was determined to begin when the point-to-point variation was greater than 2dB. Once the point-to-point variation was exceeded, the algorithm would look for the end of the spike as determined by the signal returning to within the 2dB of the start point. If the size of the spike was shorter than a maximum width, 30 points for example, the spike was replaced by a line connecting the ends of the spike. In this way it was believed that data loss would be kept to a minimum since every spike would ideally be replaced by data on the tune-curve. The result was much noisier than the raw data fit, as seen in *Figure 31*. The typically poor fits, examples of which are shown in *Figure 32*, demonstrated that the spike stripping not only failed to eliminate the spikes, but when the spikes were expressed regularly at 60Hz intervals the spike stripping algorithm incorporated the spikes. The spikes were difficult to eliminate completely because the spacing between the spikes was sometimes irregular, as shown in *Figure 21*, and local minima were plentiful across the sampled tune range.

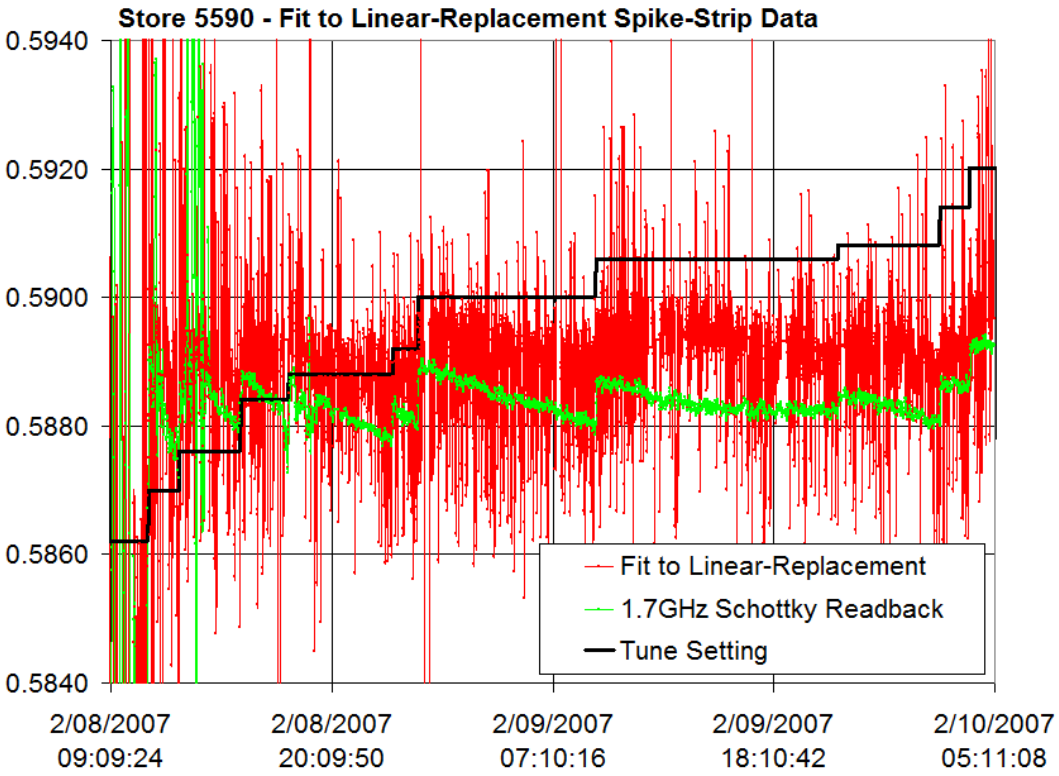

Figure 31 – The Tune parameters after employing Omission Spike-Stripping and fitting to the spike-stripped data using LMA as compared to the 1.7GHz Schottky and the tune settings. This is much noisier than the fit to the Raw Data. Note that the scale has been increased from *Figure 30* to fit the data on the chart, but it still generally follows the datalogged tune from the 1.7GHz Schottky.

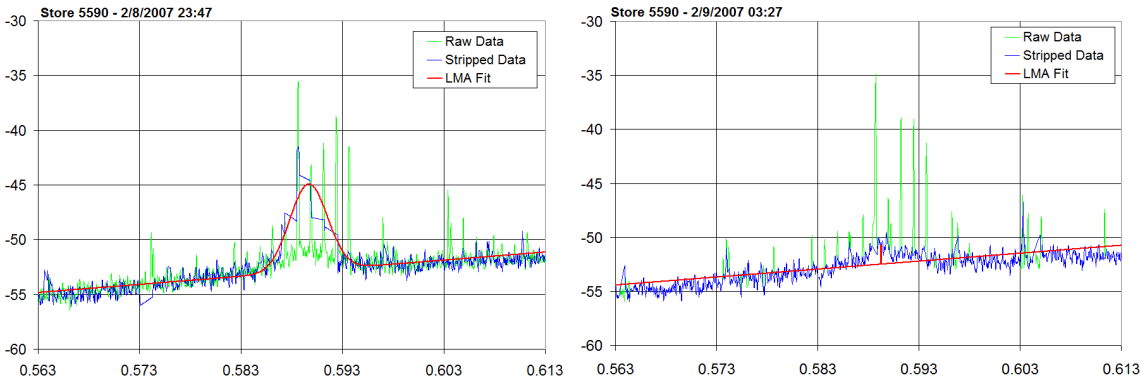

Figure 32 - Two plots of fits to linear-replacement spike-stripped data. Both are fairly common within the dataset. The red trace on each plot is the fit to the spike-stripped data, shown in blue. The raw data is shown in green for comparison. The errors in spike stripping and poor fits result in the noise of the measured tune seen in *Figure 31*.

The ‘Omission’ spike-stripping method was the same as the ‘Linear-Replacement’ method except that the data determined to be spikes was omitted entirely. This created a number of holes within the data set but it was hoped that these holes would reduce the effect that the spikes had on the LMA fit. The datasets from Store 5590 were fit to again, as shown in *Figure 33*. The ‘Omission’ method appeared to be better than the ‘Linear-Replacement’ method in the reduction of the noise in the fit tune, but it still suffered from poor fits due to the same poor point-to-point gradient method used for the Linear-Replacement method as seen in *Figure 32*, resulting in additional noise in the fit parameters.

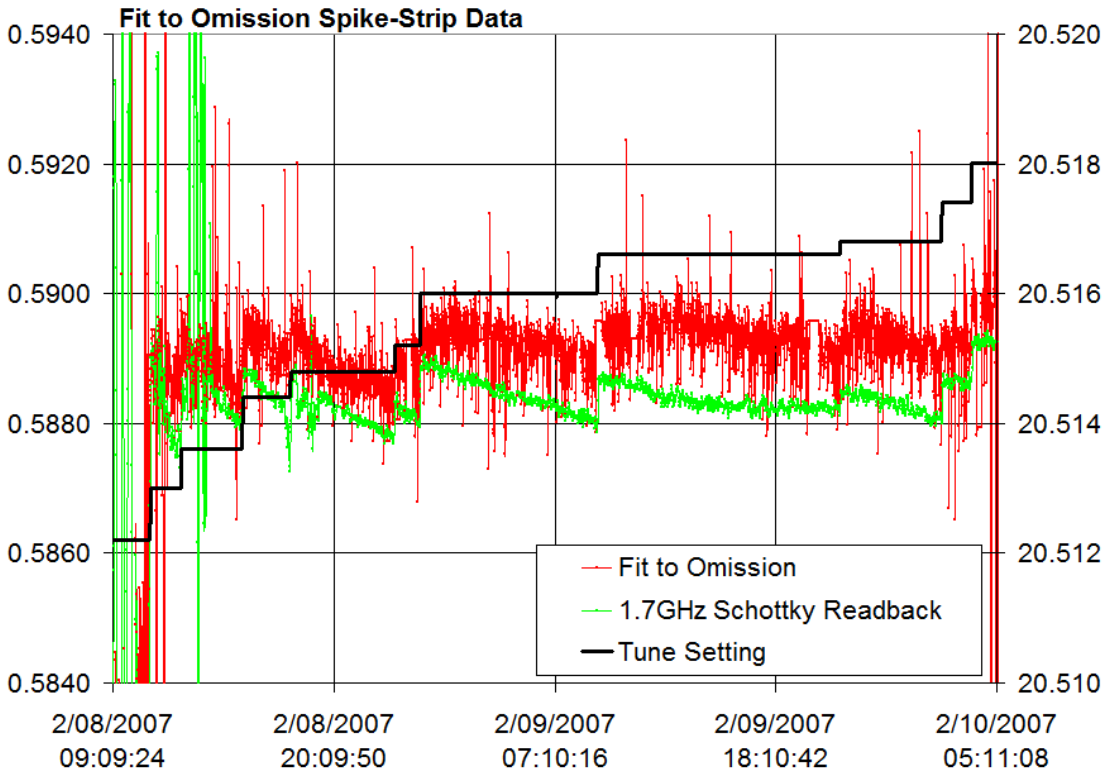

Figure 33 – The Tune parameters after employing Omission Spike-Stripping and fitting to the spike-stripped data using LMA as compared to the 1.7GHz Schottky and the tune settings.

Rather than depend on the point-to-point gradient method used in both the ‘Linear-Replacement’ and ‘Omission’ spike-stripping methods, the ‘Fit-Threshold’ method determined a spike based on an LMA fit to the raw data, which yielded a fit like the one in *Figure 25*. While the fit parameters were a bit unreliable as discussed earlier,

they also tended to remain close to the bulk of the pertinent data. A threshold of 2dB was set, effectively adding to the linear offset parameter, p_5 from *Equation 42*, and data above this new curve was omitted. Anything below the threshold curve was fit to again using LMA using the terminal fit parameters from the first fit. The result was a progression similar to the one shown in *Figure 34*. Fitting the remainder of the data from store 5590 in this fashion, shown in *Figure 35*, resulted in much less noise than either the 'Linear-Replacement' or 'Omission' methods previously applied.

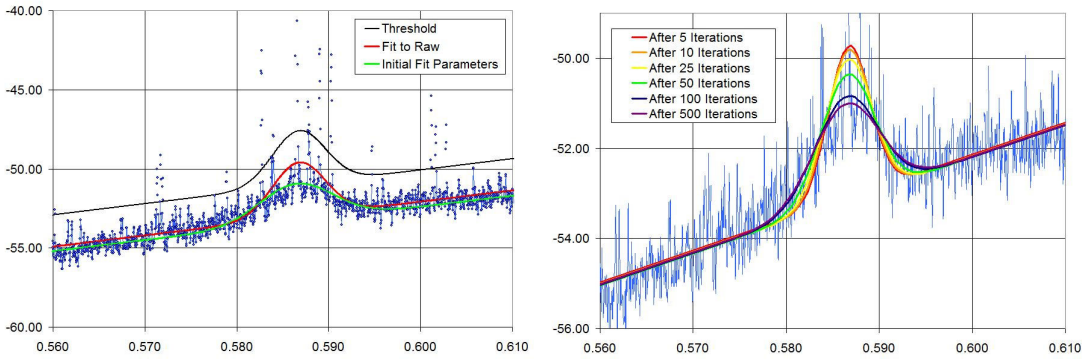

Figure 34 – Shown above are the two parts of the Fit-Threshold Spike-Stripping method. On the left a threshold is established by fitting directly to the raw data, as in *Figure 25*. A threshold, the black trace, is established by adding 2dB to the raw fit. The data above the threshold is ignored and the remaining data is refitted. The figure to the right shows the refit over 500 fit iterations in 5-iteration intervals.

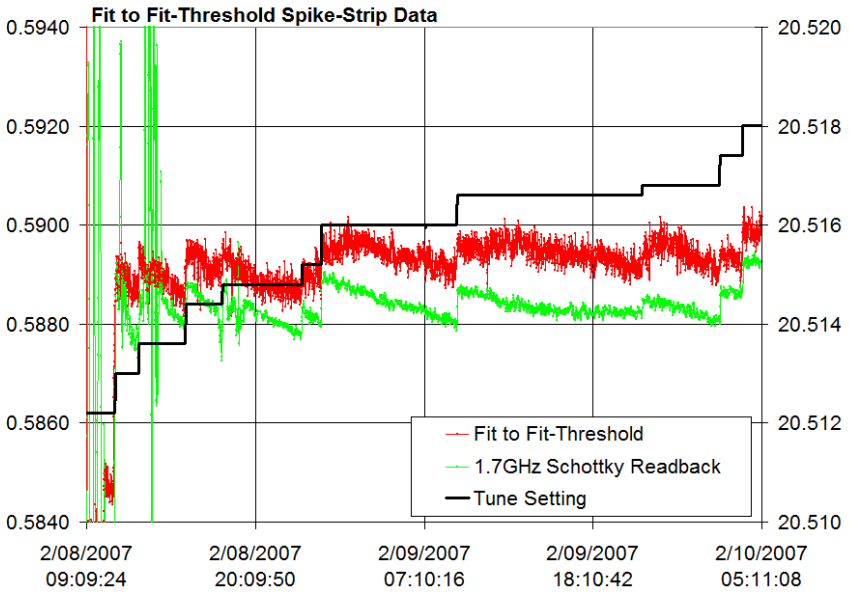

Figure 35 – The Fit-Threshold Spike Stripping method on Store 5590. There is still some discrepancy between the 1.7GHz Schottky and BBQ tune measurements on the order of 0.001, but this may be due to tune coupling.

The Fit-Threshold method also followed the tune readbacks from the 1.7GHz Schottky more closely than the fit to the raw data, but not by much as the difference between the two was generally less than 0.001 tune units for most of the store, as seen in *Figure 36*. In spite of this small overall difference, the spikes must be eliminated to provide the most precise tune measurement and so the Fit-Threshold method is clearly the method of choice.

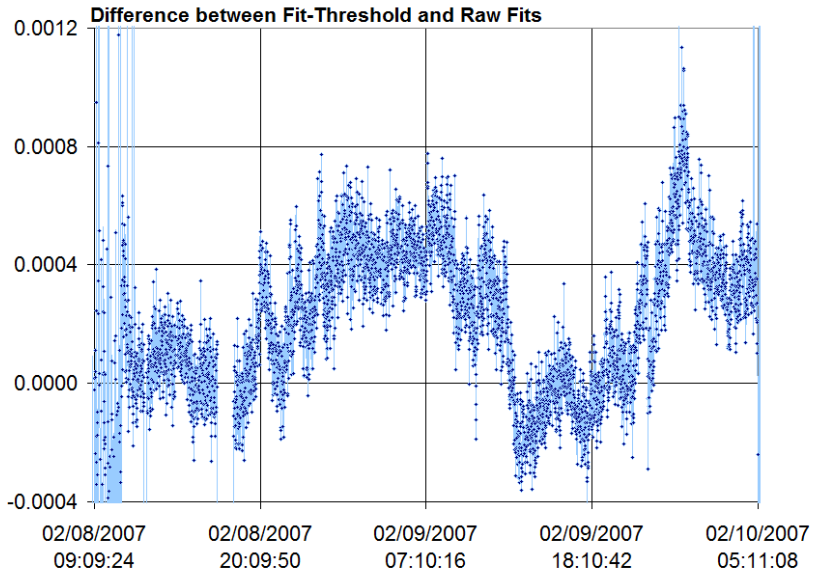

Figure 36 – This plot shows the difference between the Fit-Threshold tunes and the Raw Fit tunes.

The fit parameters used for each of these methods were the same and are summarized in *Table 5*. They originated from a fit to a test data set, like the ones shown in *figures 25* and *34*, but these are not necessarily representative of all of the data. This becomes apparent when examining the tunes at the beginning of Store 5590 before ramping from 150GeV to 980GeV. At that point, the tune Gaussians were much larger, as seen in *Figure 37*, and the parameters used to fit to the tunes later in the store were not optimal for fitting. As seen with the test data sets in *Figures 27, 28 and 29*, this results in noise in the fit. However, by tweaking the parameters, a smooth transition could be found where the tune was satisfactorily close up the ramp and into the store as seen in *Figure 38*.

Table 5 – Beginning of Store and In-Store parameters used for analysis of Store 5590

		Beginning of Store Values	In-Store Values
Gaussian Amplitude	p_1	10	1.86
Tune	p_2	0.581	0.588
Standard Deviation (σ)	p_3	0.0103	0.0026
Linear Slope	p_4	71.677	75.502
Linear Offset	p_5	-95.239	-96.173

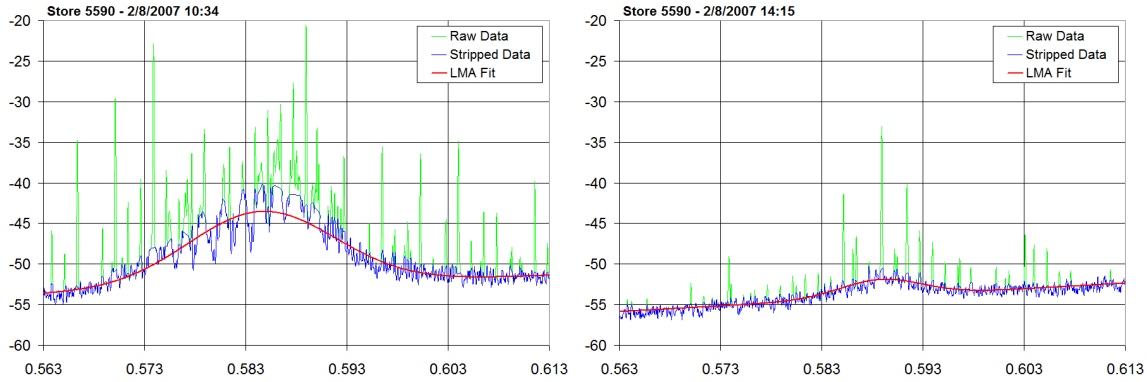

Figure 37 – Two tune traces from Store 5590 taken while at 150GeV (left), and the other while at 980GeV (right). Again, the raw data is shown in green, the spike stripped data in blue and the fit in red.

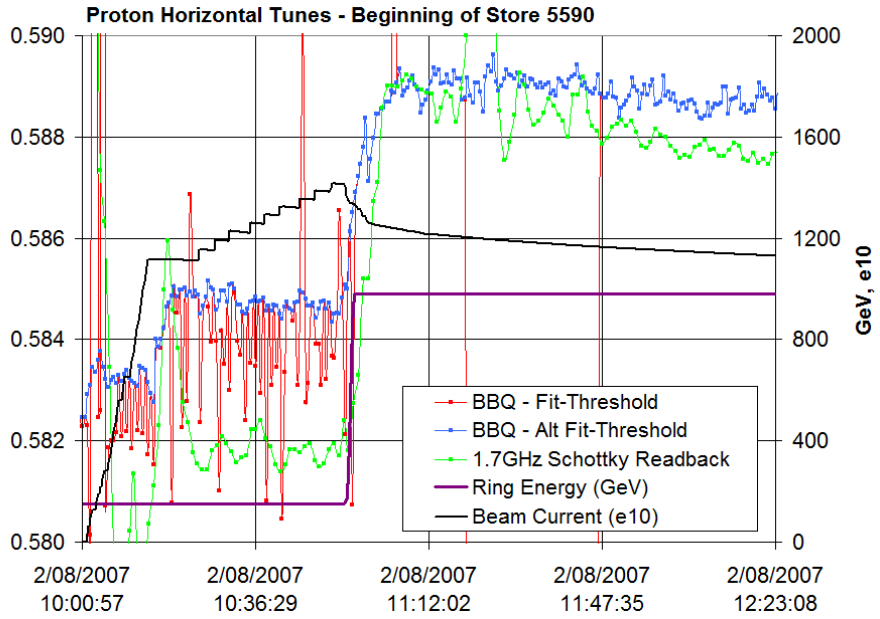

Figure 38 – The default parameters used for in-store analysis failed to fit well at 150GeV. Tweaking the parameters, however, resulted in a much better fit that was carried through into the store by feeding forward appropriate fit parameters, as shown by the alternate parameter fit (blue).

It should be noted that the data examined thus far is for the horizontal plane, but similar results were obtained for the vertical plane. One interesting trend presents itself when examining the horizontal tune against the vertical tune changes. As one would expect, the largest tune movements correspond to the horizontal tune adjustments. There are also minor movements that appear to correspond to the vertical tune adjustments as shown in *Figure 39*. This suggests that coupling of the tunes is present, but further investigation is required. The ability to measure coupled tunes would prove advantageous to the BBQ system since the 1.7GHz Schottky system exactly cancels coupling effects.[xiii]

The BBQ tune measurement system was then set up to look for Horizontal Antiproton tunes. The signal from the antiprotons is relatively small signal because the antiproton intensity is lower than that of the protons by a factor of 10. As a result the tunes were difficult to see for the duration of the store, as seen in *Figure 11*. Given an upper average antiproton bunch intensity of $85 \cdot 10^9$ and a 980 GeV bunch length of 1.7 ns would yield an antiproton bunch current, J_B^- from *Equation 38*, of 50 EHz, which is well below the 93 EHz threshold bunch current measured for the protons. This would suggest that no attenuation should be required to see the antiproton tunes at the beginning of the store. This was difficult to determine from the data since the tune bump for the antiprotons was never as prominent as that of the protons. As for filtering prior to the diode boxes, it was expected that they would reduce the bandwidth of the input signal, thereby improving the quality of the signal following the diode boxes, but they would also serve to attenuate the signal to some extent. The obvious question is whether the 70 MHz low pass filters would help or hurt tune measurement.

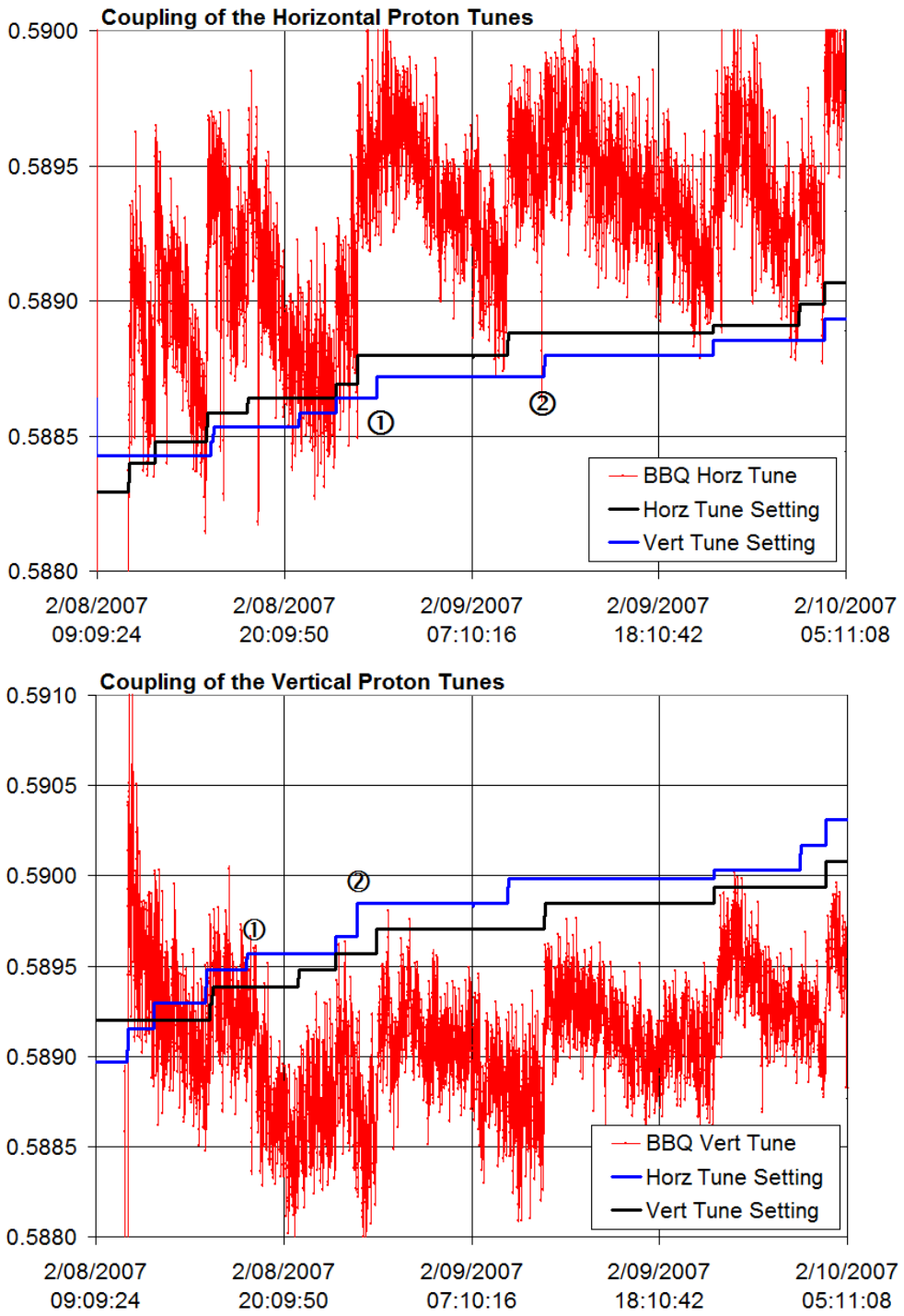

Figure 39 – The horizontal tunes, shown above, and vertical tunes, shown below, are generally moved by adjustments to the tune setting, but there were a couple instances where the vertical tune in the plane in question appeared to move with the opposite plane's tune setting, denoted as ① and ② in each plane. This indicates coupling.

Two scenarios were tried over the period of several stores. The first scenario involved no filtering and the other employed 70MHz low pass filters before the diode boxes, as had been done with the horizontal proton tune measurement. One store without filtering, 6389, and one with filtering, 6413, were picked for analysis due to their similarity. Each store collided for roughly 17 hours with similar initial and final bunch currents, and similar sets of tune changes were made. The process of analysis was similar to the one performed when determining optimal initial fit parameters for the protons. First the data was surveyed for a file that showed signs of a tune bump, which was found at 15:51:53 on 9/11/2008 during store 6413. As with the sample proton fit in *Figure 25*, a fit was performed to the dataset by hand to approximate initial tune values for use as initial fit parameters in an LMA analysis of the single trace. A round of fitting resulted in similar values, as shown in *Figure 40*, which were used for the initial parameters in the LMA analysis for the entire set of traces saved from both stores 6389 and 6413. The hand and post-single-fit parameters for use in Stores 6389 and 6413 are listed in *Table 6*.

Table 6 – Hand and Post-Single-Fit parameters for Antiproton Tune Store analysis

		Hand Fit Parameters	Parameters After Single LMA
Gaussian Amplitude	p_1	1.5	1.204
Tune	p_2	0.5882	0.58786
Standard Deviation (σ)	p_3	0.0023	0.00188
Linear Slope	p_4	38	30.72
Linear Offset	p_5	-70.1	-65.81

The results of the analysis are plotted in *Figure 41*. At first glance the unfiltered data from store 6389 may appear to result in a better fit than for store 6413, but it lacks any motion to suggest that the tune is changing with the tune setting changes. On the other hand, there is little evidence that the fit to data from store 6413 fared any better. Other parameters, namely the standard deviation, or p_3 from *Equation 42*, were then compared between stores as shown in *Figure 42*. Store 6389 shows most of the fit standard deviations around 0.01 sporadically diving lower. Store 6413 shows a standard deviation that starts around 0.01 but then moves toward 0.001 before the readback gets noisy and finally jumps up to 0.1.

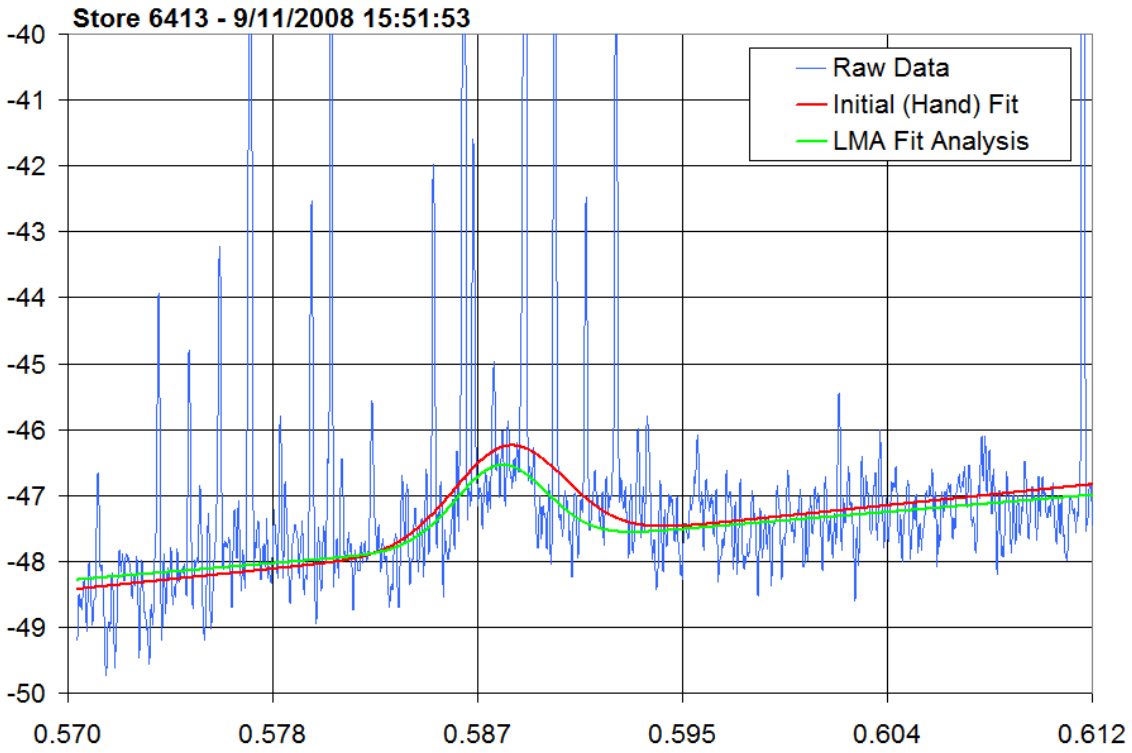

Figure 40 – Data from the VSA shows what appears to be a tune bump on the signal from the horizontal antiproton pickups at 15:51:53 on 9/11/2008 during store 6413.

As shown in *Figure 43*, a tune Gaussian with standard deviation of 0.01 will fill the entire tune space of the VSA. The tune Gaussian seen with the proton tunes and the sample antiproton tune, in *Figure 40*, is closer to 0.001. The noise spikes are on the order of 0.0001 or less. As mentioned at the introduction of the fit model, the baseline is not actually a line, but it appears that the Gaussian fit to the data from Store 6389 has combined the tune Gaussian with the curve of the baseline to return the fit parameters for a much broader Gaussian. The standard deviations from the fits to the data from Store 6413, on the other hand, start out broader and approach 0.001- the expected standard deviation of the tune bump. After a certain point, however, the standard deviation jumps up to between 0.1 and 1, which appears relatively flat across the tune span of the VSA. At that point, the tune bump can be said to have disappeared completely according to the LMA for the given fit parameters and hardware.

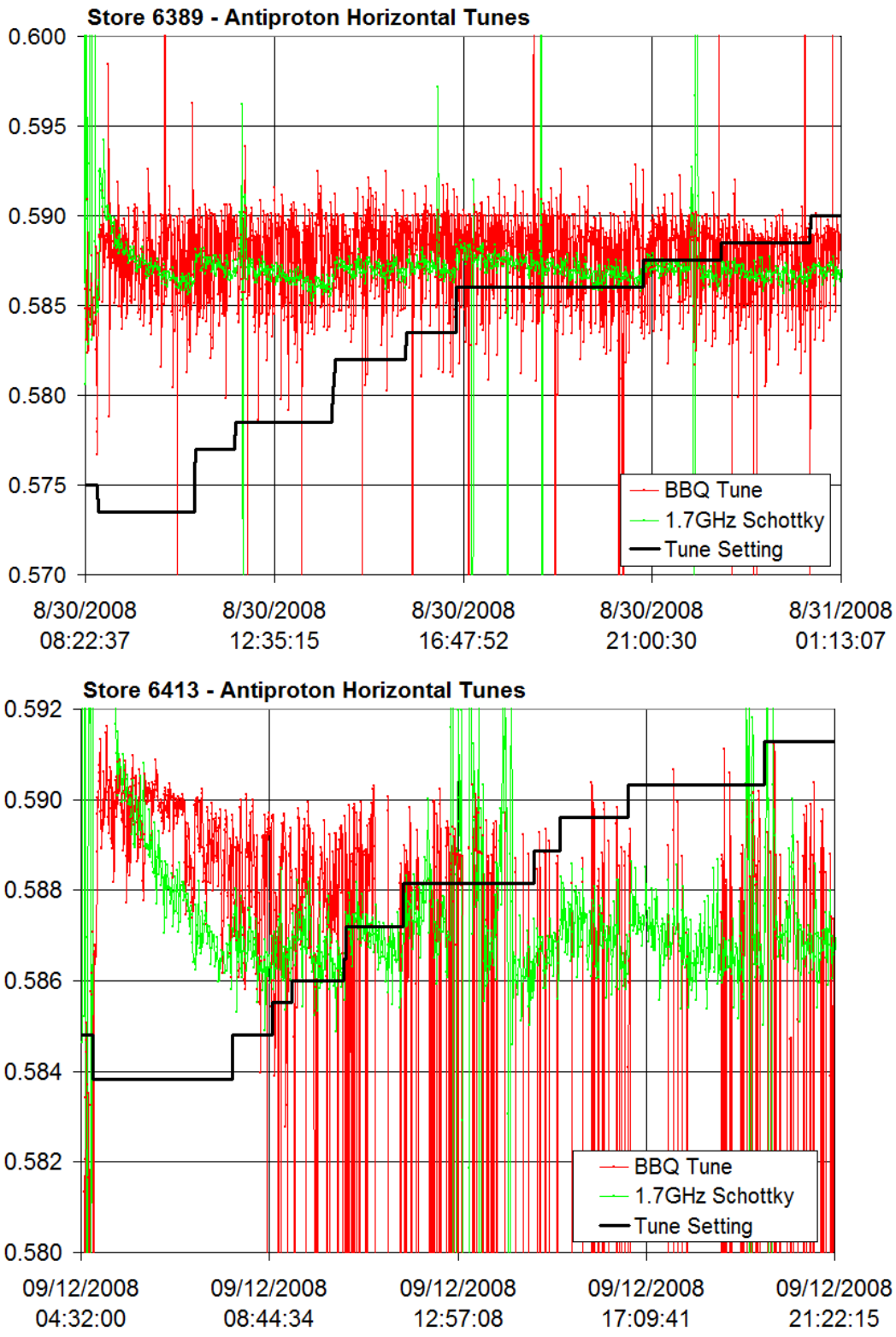

Figure 41 – BBQ fit tunes for Stores 6389 (top) and 6413 (bottom). The tune setting changes in both are shown as black traces on top of the 1.7GHz Schottky, shown in green, and the LMA tune fit parameter, shown in orange.

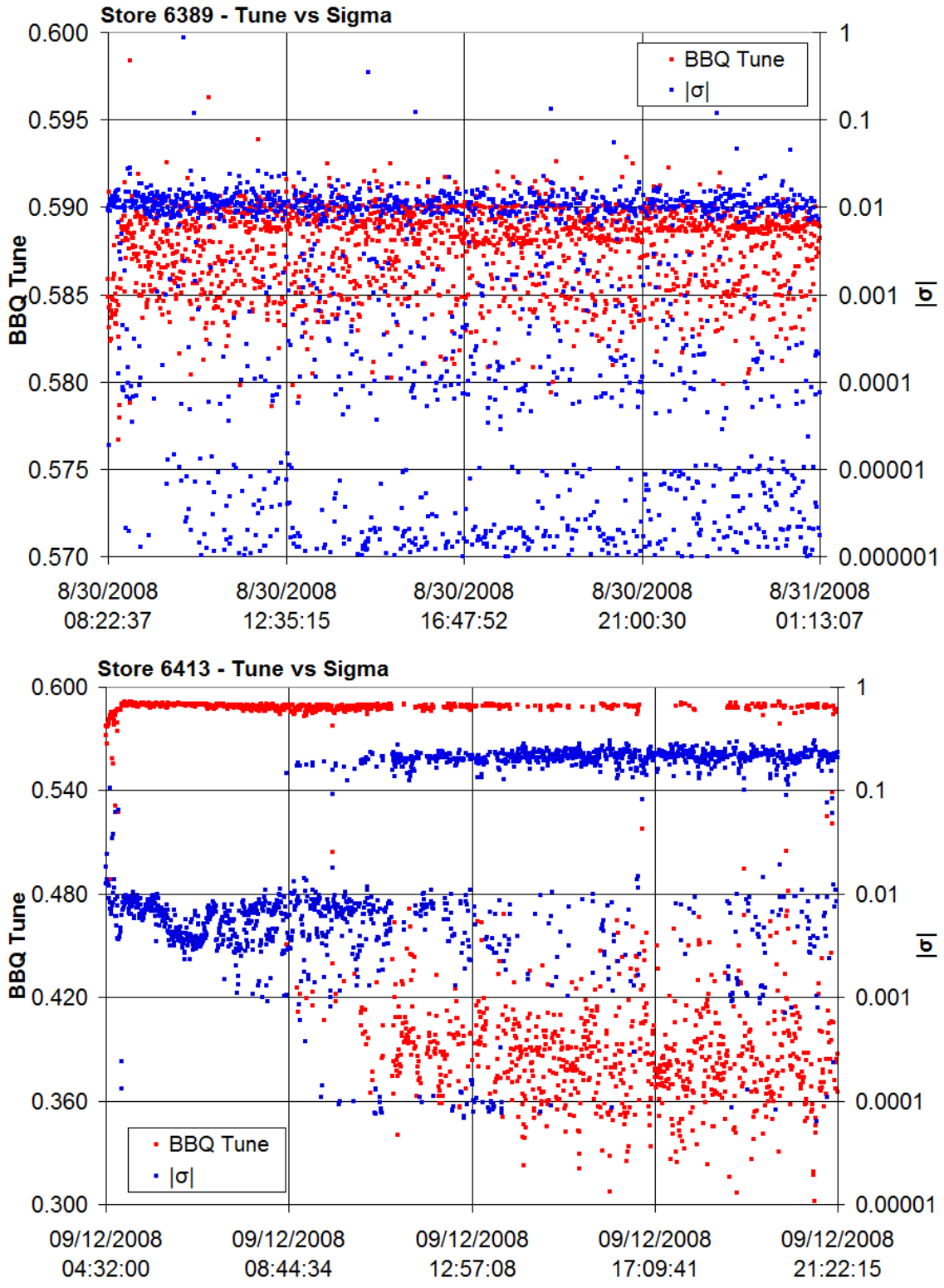

Figure 42 – Antiproton Tunes and matching standard deviations from Stores 6389 (above) and 6413 (below).

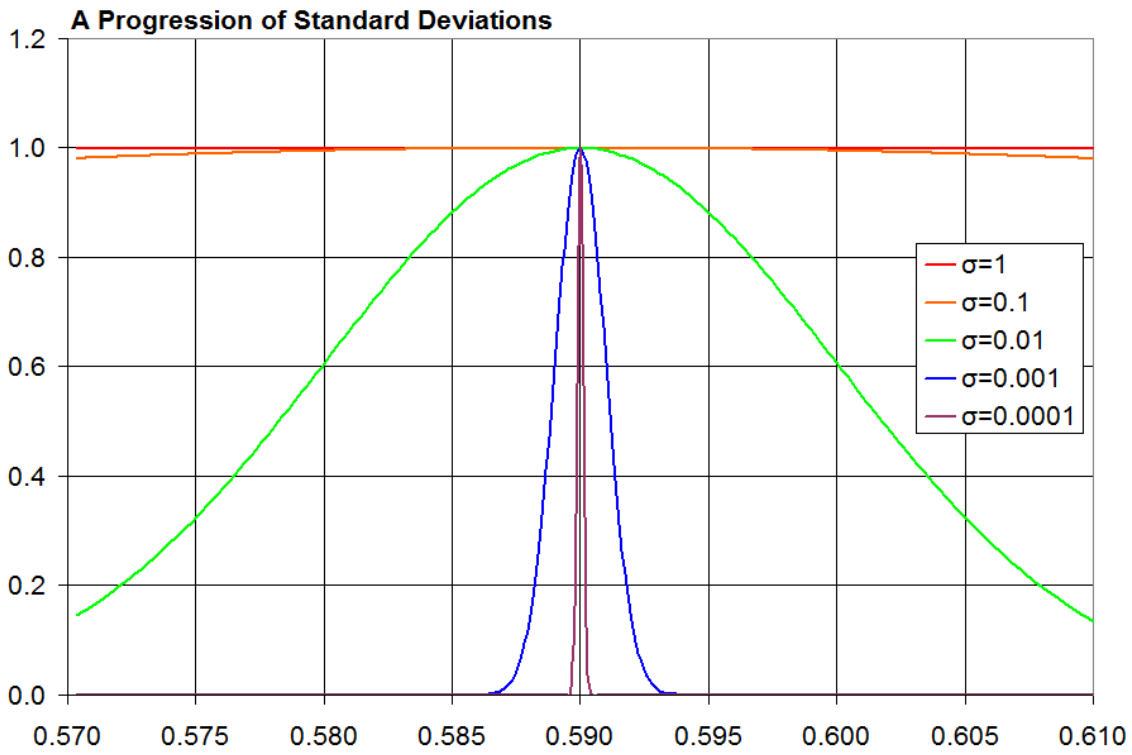

Figure 43 – A progression of standard deviations from 0.0001 to 1 across the tune span used in the antiproton tune analysis. The Gaussian amplitude has been set to 1, the Gaussian tune has been centered at 0.590, and linear components have been ignored.

Any tune measurement hardware will have a lower threshold at which point the tune is no longer measurable above the noise floor. In the case of the BBQ system, there are five variables that must be accounted for. The first four consist of hardware components: the pickups, diode detectors, front end, and VSA. The last, of course, is the fit algorithm or, more specifically, the initial fit parameters passed to the fit algorithm. If any of these pieces are not optimized to some degree, the result will decrease the accuracy of the measurement, as has been covered for each piece. An upper threshold, J_{thr}, has already been explored with respect to the proton tunes, but with the antiproton signals, the problem seems to be too little signal rather than too much. Again, the signal strength can be related to the antiproton bunch current, J_B^- introduced in *Equation 38*. Assuming that the tune measurements from the beginning of Store 6413 were following a tune bump, based on the behavior of the standard deviation parameters from the fits, a minimum bunch current can be found to be roughly 39 EHz, as seen in *Figure 44*. This

will change when the initial parameters, VSA parameters, such as averaging, the front end filtering and gain settings, or even presumably the diode detectors are changed. These settings are much more critical for a lower threshold since amplification of the signal also generally increases the noise. It may be possible to add a preamp to boost the signal prior to the diode detectors, but this was not tried.

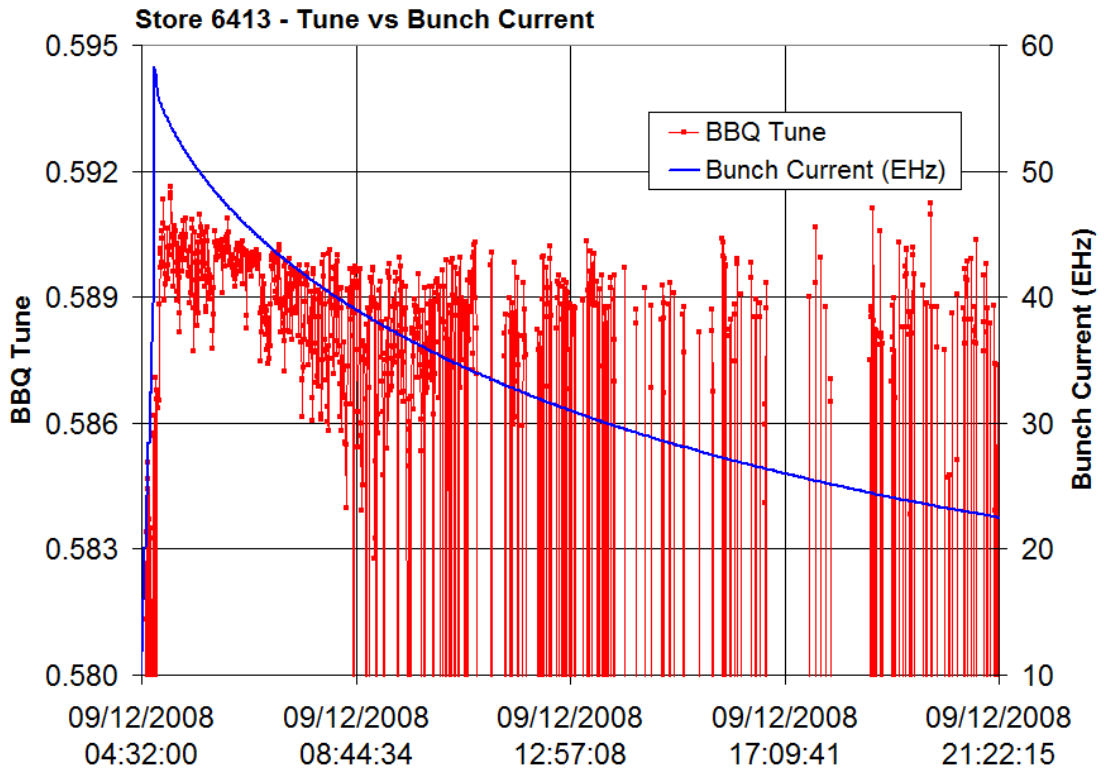

Figure 44 – The tune data from Store 6413 against the bunch current. With the given hardware, averaging, initial fit parameters, and beam conditions the tune appears to become unreadable just under 40EHz.

One alternative is to increase the averaging in the VSA which will help average out random noise and hopefully help resolve the tune bump. This could pose some problems if the averaging is turned up too high, both in terms of taking too long to detect shifting tune and incorporating undesired data. The measurement time, or the time for the VSA to sweep the frequency range once, is on the order of 1 second in the nominal setup that has been used for tune measurement for the duration of this paper. Averaging over 600 measurements, would therefore take about 5 minutes. While the 60 Hz noise does not

move very quickly, it does move, as seen in *Figure 22*, which could result in an offset in the tune from the LMA fit. It also stands to make a very narrow noise spike much wider and, therefore, a much more likely target for LMA as shown in *Figure 29*.

The analysis of Store 5590 showed that measurement of the proton tunes was best achieved using the fit-threshold spike stripping method. It also demonstrated the need to be able to easily manage the initial fit parameters for the LMA. In contrast, the antiproton tunes proved more difficult to measure. Using the fit-threshold spike stripping method on the horizontal antiproton tunes showed some promise in Store 6413, but the vertical antiproton tunes could not be measured. Furthermore the measurements of the horizontal antiproton tunes suggest that lowering the threshold bunch current, J_{thr}, would result in a better tune measurement and may even allow for measurement of the vertical antiproton tunes.

The BBQ Tune Measurement Program

The methods and analysis of the data from Stores 5590 and 6413, for the proton and antiproton tunes respectively, were performed offline. The next challenge was to develop a program to give live, real-time readbacks of the tunes. Because the LMA interface was stand-alone and written in C++, it was portable to the Fermilab accelerator controls system, ACNET. This was done in the form of a user library, making it accessible to a number of programs.[xiv] The BBQ Tune Measurement program, or PA4040 as it is known to the ACNET controls system, was developed with three basic guidelines. First, the desired program had to be able to collect spectra from the BBQ VSA and perform fits to the data at a rate on the order of 1 Hz to make it competitive with other tune measurement systems in the Tevatron. Additionally, a means to control the initial fit parameters, whether the resulting fit parameters were fed forward to the next round of fitting, and other basic control of the program was desired. Finally, it was desired that the program run as a background process to help prevent accidental termination.

The speed of the data collection and the Levenberg-Marquardt analysis that followed is generally dominated by the analysis, or more specifically, by the number of iterations required to close in on a satisfactory fit. The maximum number of fits was set to 200, which resulted in an upper limit if no adequate fit were readily found. In using the fit-threshold spike stripping method, previously described, recall that two rounds of LMA are performed. The first is used to set the threshold to eliminate the spikes. Once the spikes are eliminated, a second round of LMA is used to measure the tune.

To provide control and readback interaction with ACNET, ACNET parameters were employed. Each parameter can be defined with a number of components, such as analog and digital settings and readbacks as well as alarm blocks for the digital and analog readbacks. The readback components can be datalogged, which is to say that their values can be stored at a settable rate or on a clock event by a tertiary process. In the

interest of monitoring the fit quality, parameters were also made to output the number of points kept after the fit-threshold spike stripping and the fit error, the sum of the squares of the differences between the fit and the data, after the final LMA fit.

Parameters were also made for an average tune, $\overline{q}$. On the first measurement loop the average will be set to the tune, but each subsequent measurement will only change the average tune by a fraction. The result is a weighted average of all previous tune measurements and the current tune measurement of the form

$$\overline{q}_{new} = \gamma \cdot p_2 + (1-\gamma) \cdot \overline{q}_{old} \tag{58}$$

where γ is the averaging factor and p_2 is the tune fit parameter from *Equation 42*.

Assume, for a moment, an initial tune of $p_2 = q_0$. If there were an abrupt tune change, such that $p_2 = q_1$, the average tune would start changing with each measurement step, as shown in *Figure 45*. The result is an approximately exponential approach of the average tune toward q_1 such that,

$$\overline{q}_{new} = q_0 + \left[(q_1 - q_0) \cdot \left(1 - e^{-n \cdot \gamma}\right)\right] \tag{59}$$

for each measurement step, n. The averaging factor was set to $\gamma = 0.02$, resulting in half the tune change $q_1 - p_2$ being averaged roughly every 35 steps. If the measurements are taken on the order of 1Hz, the average tune will be constantly exponentially approaching the current tune roughly over the period of a couple minutes.

The averaging is intentionally performed before the question of whether or not to feed-forward the fit parameters is addressed so that it will incorporate even poor fits. In this way it can also serve as a measure of quality in the measurement. If fits are made to data, the average tune parameter will follow the datalogged tune parameter whereas deviation will indicate places that the program had trouble fitting to the data. This may change in the future since more direct methods of determining the quality of the fit exist.

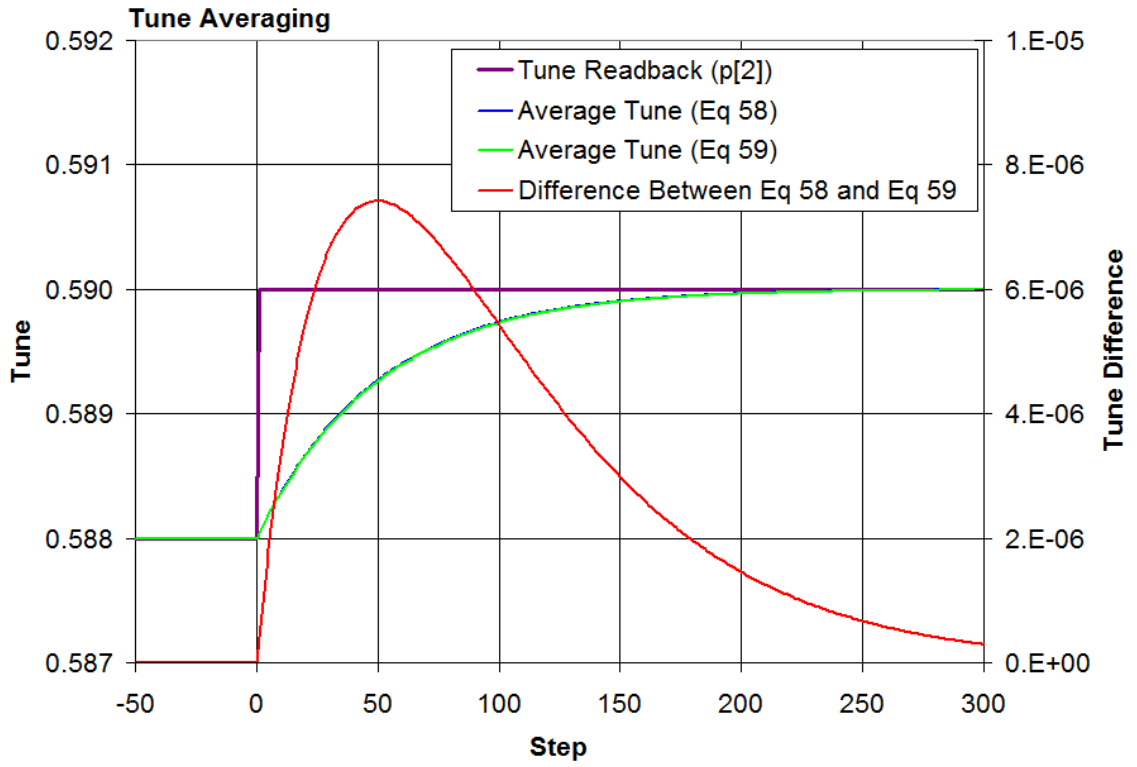

Figure 45 – An example of tune averaging over a discrete step from 0.588 to 0.590. There is a small difference between *Equations 58* and *59*, but the approach to 0.590 is roughly exponential.

The analog alarm block contains properties that can be interpreted as min/max or nominal/tolerance pairs. It was decided to use the nominal settings as the initial fit parameters while the tolerance settings were used as feed-forward conditions. If each of the parameters was within the set tolerance of the nominal setting the fit parameters, p_x, from that round of LMA fitting would be used as the initial fit parameters for the next round of fitting, allowing for a quicker and better fit while keeping some constraints on the fit to keep it from wandering too far. On the other hand, if the fit parameters were out of tolerance, they could be set back to their nominal values prior to the next round of fitting.

General communication with the VSA was effected through a single ACNET parameter, T:BBQVSA. This parameter was given both an analog setting and digital

setting. The digital setting is Boolean and is used for turning the measurement on. When the program loads it was decided that it should start measurements without prompting. The program therefore sends a command to set the digital status of T:BBQVSA to 'on'. Once this has happened the program can be told to stop taking measurements by turning the device to 'off' from a parameter page, PA0052, or the like. The analog setting is used to send some basic commands to the VSA and communicate with the program. Setting T:BBQVSA to -1 kills the program by telling the program to return 0 from its main routine. Setting it to 1 request that the program send a single autoscale command to the VSA for both planes- a handy command if looking at the scope remotely as the SCPI commands[xv] to effect the same are

```
DISP:WIND1:TRAC:Y:AUTO ONCE;
DISP:WIND2:TRAC:Y:AUTO ONCE;
```

which can be sent through Telnet or other SCPI terminal interface. Setting T:BBQVSA to 2 or 3 will turn off or on feed-forward respectively. Setting T:BBQVSA to 4 or 5 leaves the fit parameters at their previous good values or returns them to 0 respectively. This will not have an effect on how the program works, but will have an effect on what is seen on plots of the fit parameters. If a given trace results in fit parameters that are out of tolerance they are not fed forward, but rather, they are set back to their nominals. Returning nominal parameters to the ACNET parameters is rather pointless so the program will either return each of the fit parameters for that plane to 0 or just leave them where they are. The program defaults are to feed-forward good fit parameters and to return the ACNET parameters to 0 if the fit is out of tolerance. Setting T:BBQVSA to 6 will request that the program send an ABORT command to the VSA, which causes the scope to abort and restart its current measurement. It will also take the opportunity to auto-zero and restart averaging. Finally, any other non-zero setting to T:BBQVSA will result in the program updating current nominals and tolerances for the fit parameters as well as VSA properties, namely the center frequency, span and number of measurement points across that span.

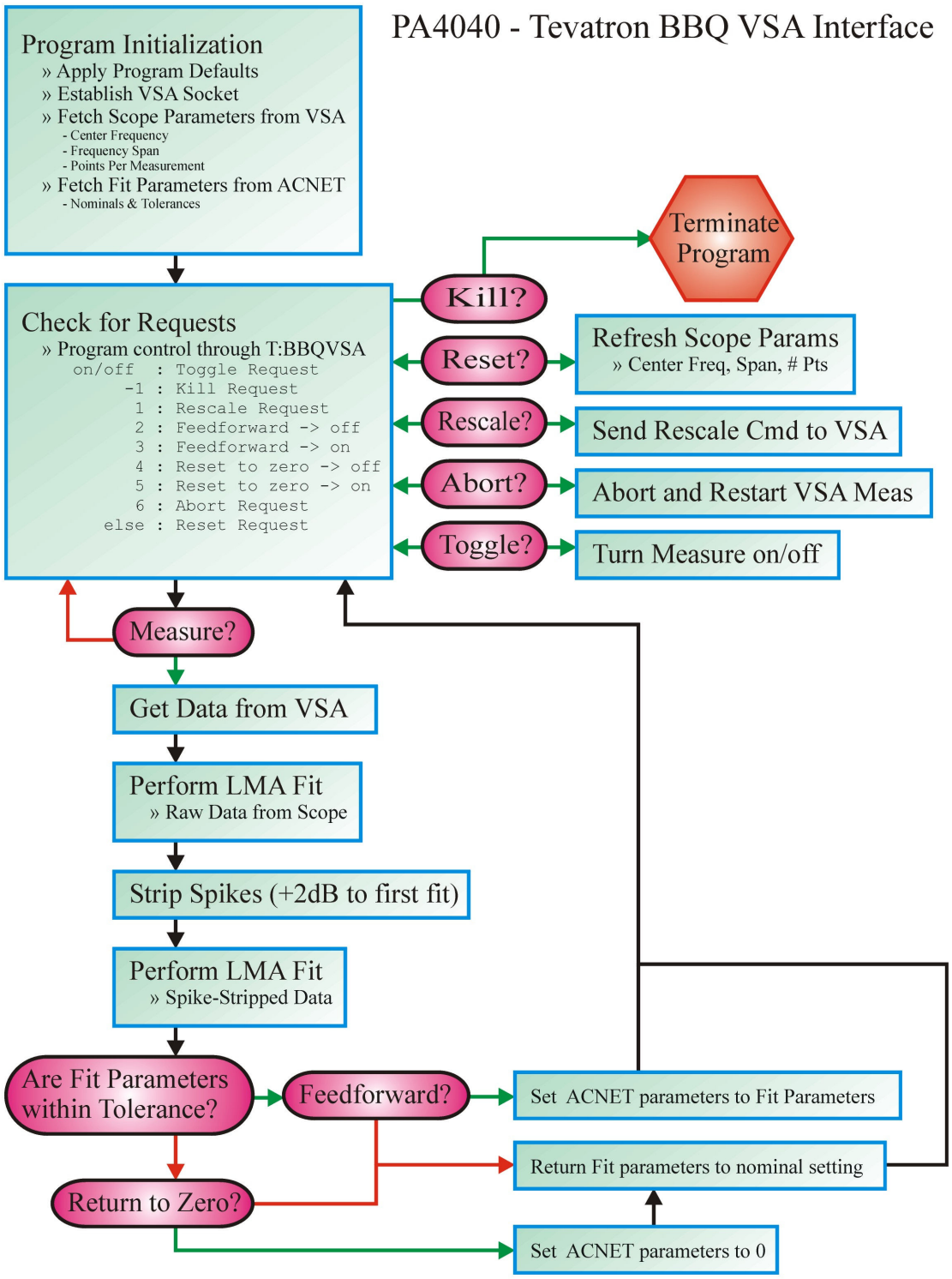

Figure 46 – A basic block diagram of the BBQ Tune Measurement Program, PA4040

Once complete, PA4040 was registered with MECCA, a code-capture system. The primary function of MECCA is to ensure programming standards and provide a buffer between the programmer and programs available to the Fermilab community through ACNET.[xvi] Once code is compiled and archived by MECCA it can be run from an ACNET console as a PA program. PA4040 was designed to be run as either a normal PA program or a slot 7 background process. PA4040 currently automatically runs as the slot 7 background process on CLX8 in the Main Control Room. A flowchart covering the operation of PA4040 is shown in *Figure 46* and the code is listed in *Appendix A*.

The fit parameters were added to 1 Hz dataloggers to record the output from PA4040 each second. As with the offline measurements of Store 5590, the BBQ system was hooked back up to the E0 proton pickups and the tune parameters were optimized for the proton tunes as shown in *Table 7*. 1ns of BNC cable was used to connect the diode boxes to the back of the BBQ front end. 3dB pads were placed on the input to the diode boxes as shown in *Figure 19*, to reduce the maximum bunch current to prevent saturation of the diode detector. Data was collected between 11/03/2008 and 11/05/2008, as shown in *Figures 47, 48* and *49*. During this period three stores, 6539, 6540, and 6541, collided.

Table 7 – BBQ Parameter nominals and tolerances for Proton Operation. An 'x' in the parameter name represents an H or V for the horizontal plane or vertical plane.

Prot	ACNET Parameter		Description	Nominal Value (H / V)	Tolerance
Fit Parameters	T:BBQxHA	p_1	Gaussian Amplitude	6 / 4.5	8
	T:BBQxHQ	p_2	Gaussian Tune	0.586 / 0.585	0.02
	T:BBQPxS	p_3	Gaussian Standard Deviation (σ)	0.01	0.008
	T:BBQPxM	p_4	Linear Slope	20	40
	T:BBQPxL	p_5	Linear Offset	-65	40
Aux Parameters	T:PxQAVG		Average Tune	0.586 / 0.585	0.02
	T:BBQPxE		LMA Fit Error	1000	1000
	T:BBQPxN		Number of Points after Spike Strip	1401	600
	T:BBQVSA		BBQ VSA & PA4040 interaction parameter		

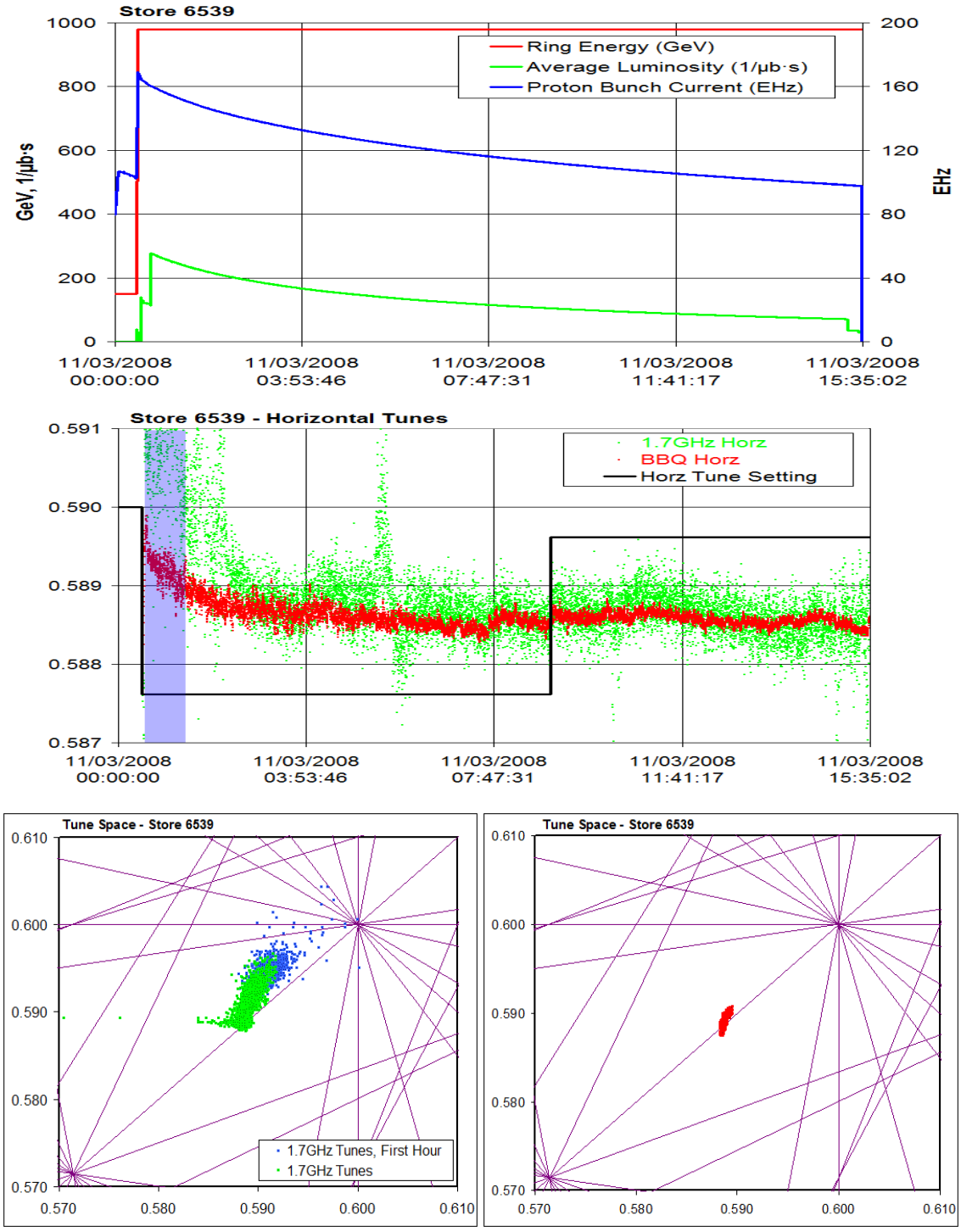

Figure 47 – A summary for Store 6539. The store was nominal size and duration as shown in the top trace. One horizontal proton tune change was made midway through the store as can be seen on the middle trace. The 1.7GHz Schottky tune-space plot can be seen on the bottom left. The BBQ tunes over the same time period are shown to the lower right.

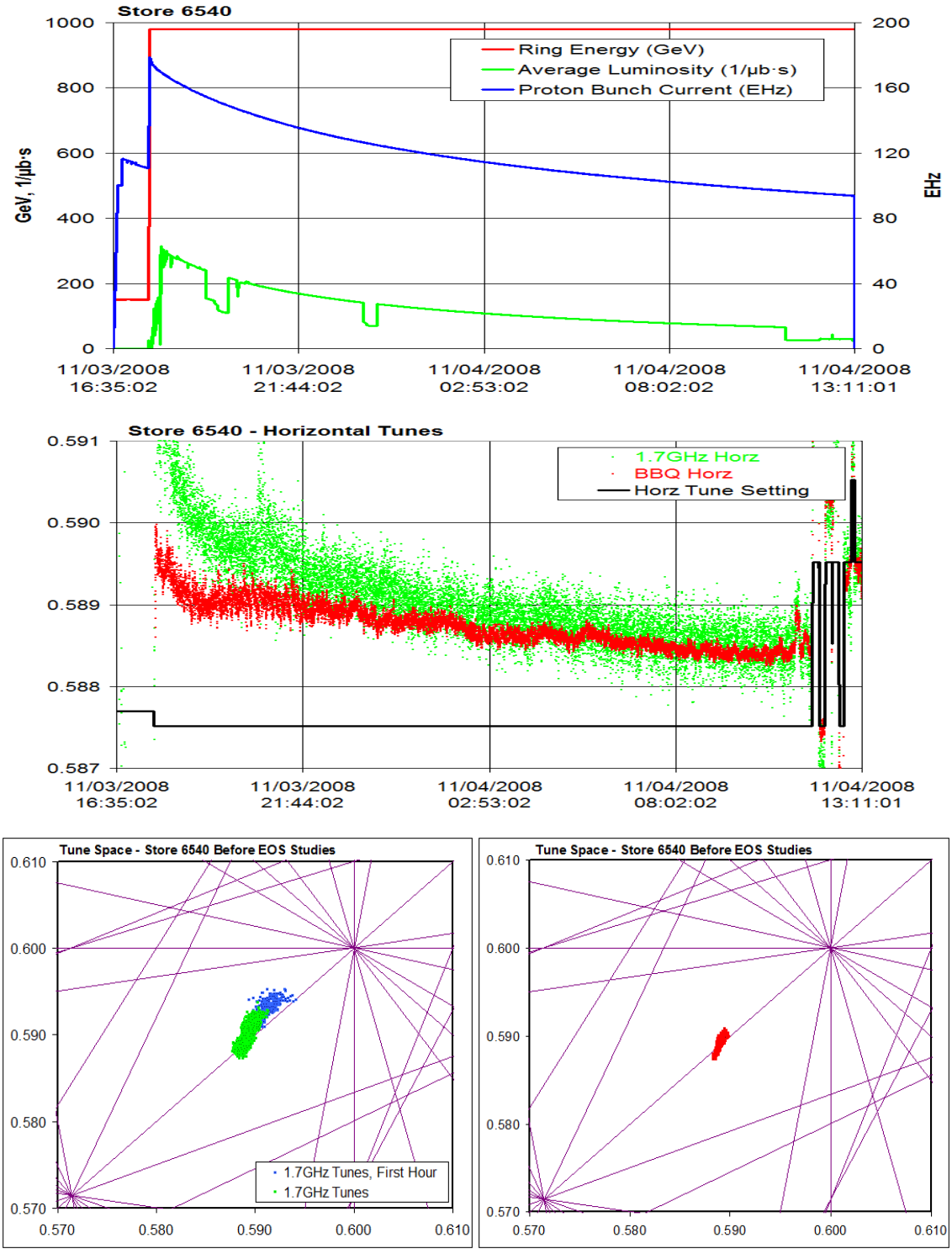

Figure 48 - A summary for Store 6540. The store was nominal size and duration as shown in the top trace. The horizontal proton tune was only changed in End of Store (EOS) Studies. The 1.7GHz Schottky and BBQ tune-space plots can be seen on the bottom left and right respectively.

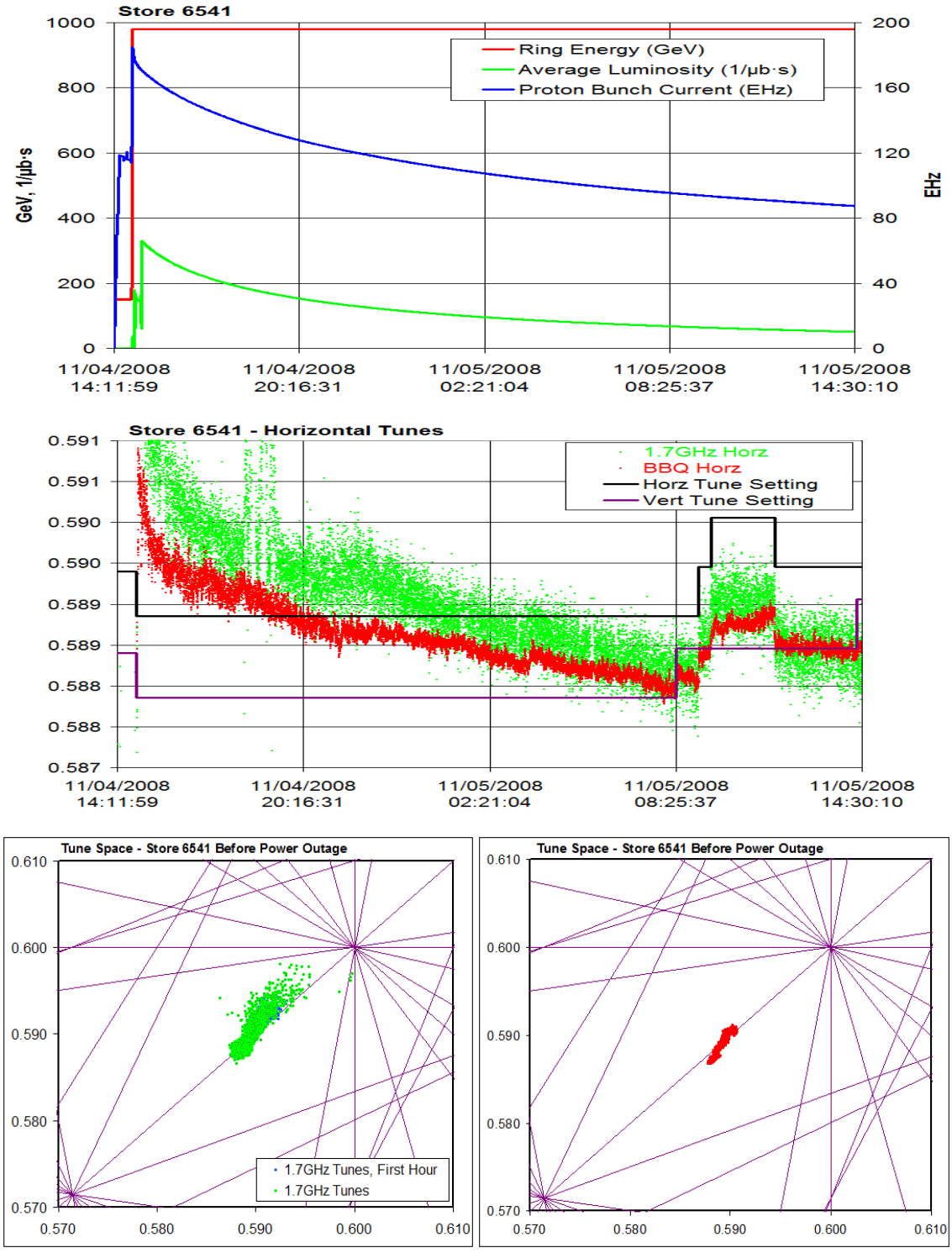

Figure 49 - A summary for Store 6541. The store was nominal size and duration as shown in the top trace, but ended with a power outage, which brought an abrupt end to the data as well as the store. Proton tunes were adjusted as needed, as shown by the middle plot. The 1.7GHz Schottky and BBQ tune-space plots can be seen on the bottom left and right respectively.

Store 6539, summarized in *Figure 47*, showcases the advantages that the BBQ system has over the 1.7GHz Schottky system, particularly at the beginning of store. The tunes from the 1.7GHz Schottky are notoriously unreliable at the beginning of the store and are generally ignored for the purposes of tuning. Even so, the horizontal tunes from the 1.7GHz Schottky are quite noisy even a couple of hours into the store, as seen in the middle trace of *Figure 47* where the first hour is delineated by the blue box. There are a number of points near the $^{3}/_{5}^{th}$ resonance line for each plane in the first hour, which would be disastrous for beam, resulting in quick beam loss. In comparison the BBQ appears to start in a much more realistic position in tune-space. It appears as though the BBQ does not suffer from the need of a one-hour moratorium on tuning the way the 1.7GHz Schottky does. One horizontal proton tune change was made midway through the store and both the 1.7GHz Schottky and BBQ systems appear to move in response.

Store 6540, summarized in *Figure 48*, featured end of store tune studies. These studies were not included in the tune-space plots, but the BBQ tunes can be seen moving in response to the tune changes. Aside from these, the tunes appear to follow the same general trends as store 6539, shown in *Figure 47*.

Store 6541, summarized in *Figure 49*, ended with a power outage that resulted in the loss of the store and an abrupt end of data. Before that point, however, the horizontal proton tune setting is adjusted several times showing correlation of the BBQ tunes with the tune setting. There is also a vertical tune change at about 08:25 on 11/5/2008 that appears to result in a small horizontal tune movement. As suggested with respect to *Figure 39*, this indicates coupling between the tune planes.

The reliability of the BBQ at the beginning of the store was used in analyzing the loss of beam near the beginning of store 6524 on 10/28/2008. The vertical proton tune setting was raised by experts, but losses eventually resulted in a quench and the store was lost as seen in Figure *50*. The cause was traced to be an accidental vertical tune change of -0.012 instead of the intended -0.0012 between stores 6521 and 6524, which can be seen on the BBQ and even the 1.7GHz Schottky to some extent.[xvii]

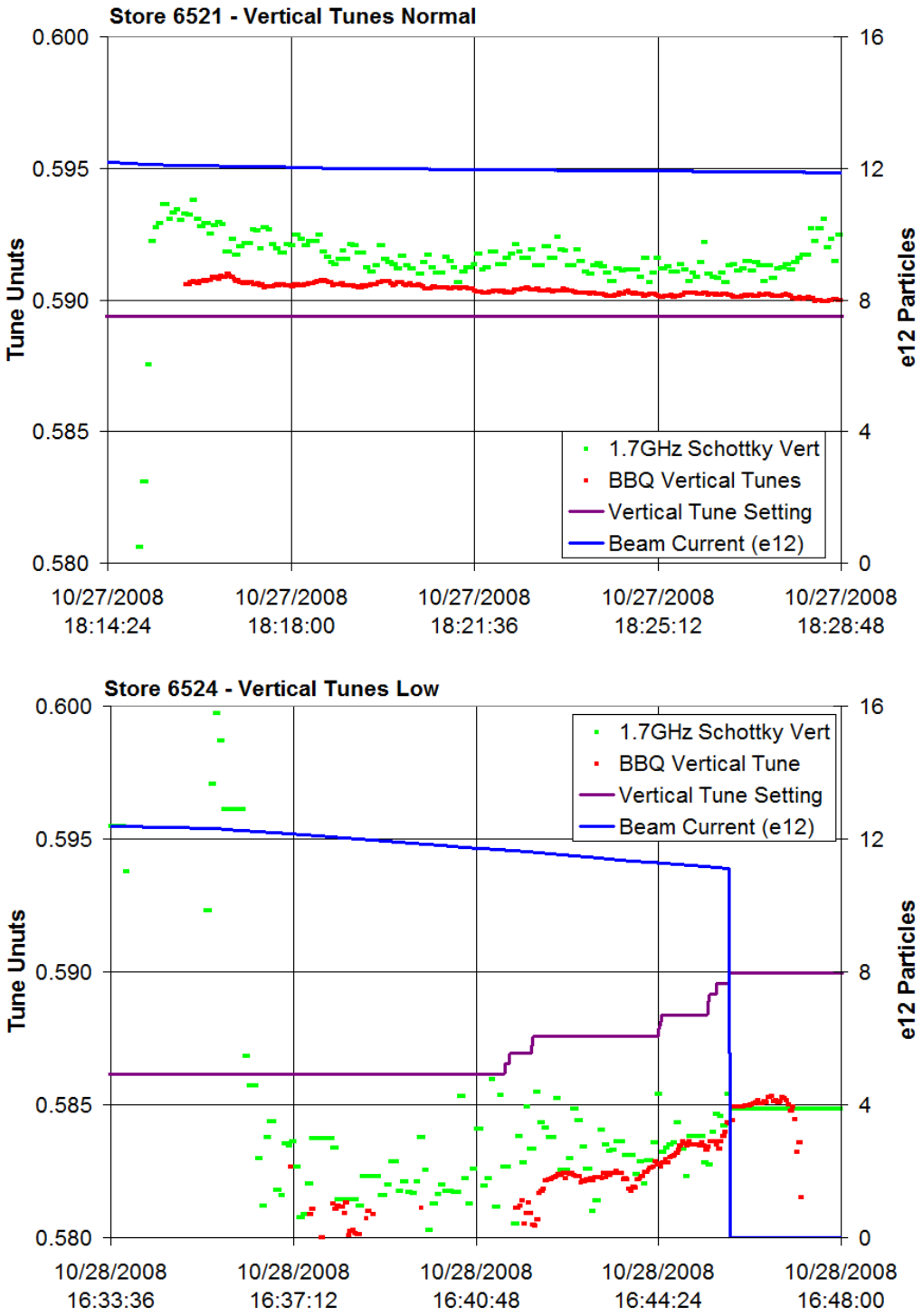

Figure 50 – The beginning of stores 6521 (above) and 6524 (below) starting at the ramp to 980GeV. Store 6524 exhibited high losses that required tuning. The vertical tunes were raised, but eventually the store was lost. The vertical tunes shown by the BBQ and even the 1.7GHz Schottky were much lower than in Store 6521, bringing them closer to the $^{7}/_{12}^{ths}$, or 0.5833, resonance line.

The BBQ was also connected to the D49 antiproton pickups and data was taken between 10/12/2008 and 10/18/2008. Adjusting VSA averaging to 60 averages, it was found that reasonable fitting could be achieved for the horizontal plane using the parameter settings in *Table 8*. During this period three stores, 6492, 6494, and 6497 collided as shown in *Figures 51*, *52* and *53* respectively.

Table 8 – BBQ Parameter nominals and tolerances used temporarily for verification of the Pbar tunes. An 'x' in the parameter name represents an H or V for the horizontal plane or vertical plane.

Pbar	ACNET Parameter		Description	Nominal Value	Tolerance
Fit Parameters	T:BBQxHA	p_1	Gaussian Amplitude	2	4
	T:BBQxHQ	p_2	Gaussian Tune	0.590	0.02
	T:BBQPxS	p_3	Gaussian Standard Deviation (σ)	0.01	0.008
	T:BBQPxM	p_4	Linear Slope	40	40
	T:BBQPxL	p_5	Linear Offset	-70	40
Aux Parameters	T:PxQAVG		Average Tune	0.590	0.02
	T:BBQPxE		LMA Fit Error	1000	1000
	T:BBQPxN		Number of Points after Spike Strip	1401	600
	T:BBQVSA		BBQ VSA & PA4040 interaction parameter		

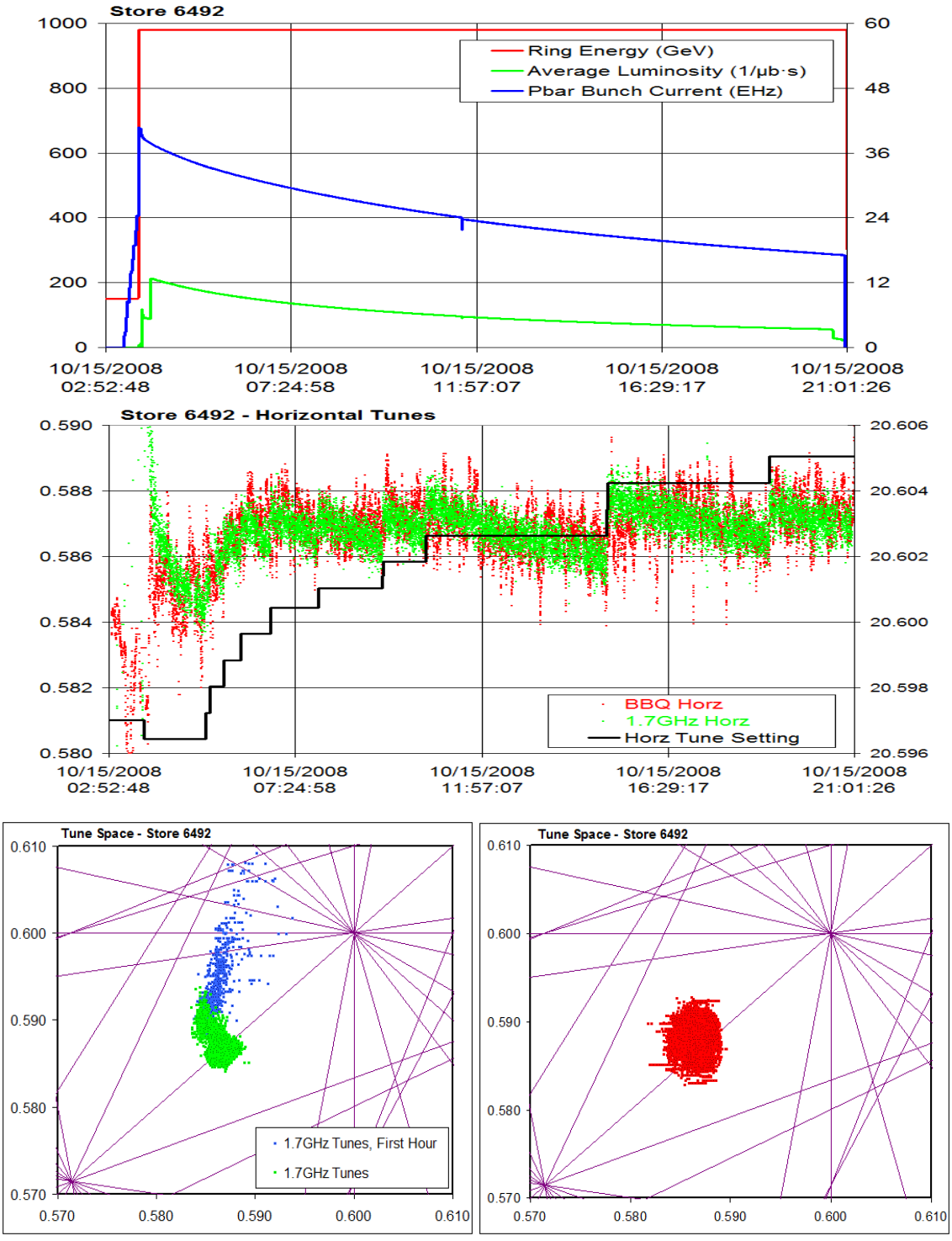

Figure 51 – A summary for Store 6492. This was during a period of smaller stores due to an instability. The store summary is located at the top. The horizontal antiproton tunes were adjusted as needed as shown in the middle plot. Tune space diagrams of 1.7GHz Schottky tunes and BBQ tunes are shown at the bottom left and right respectively.

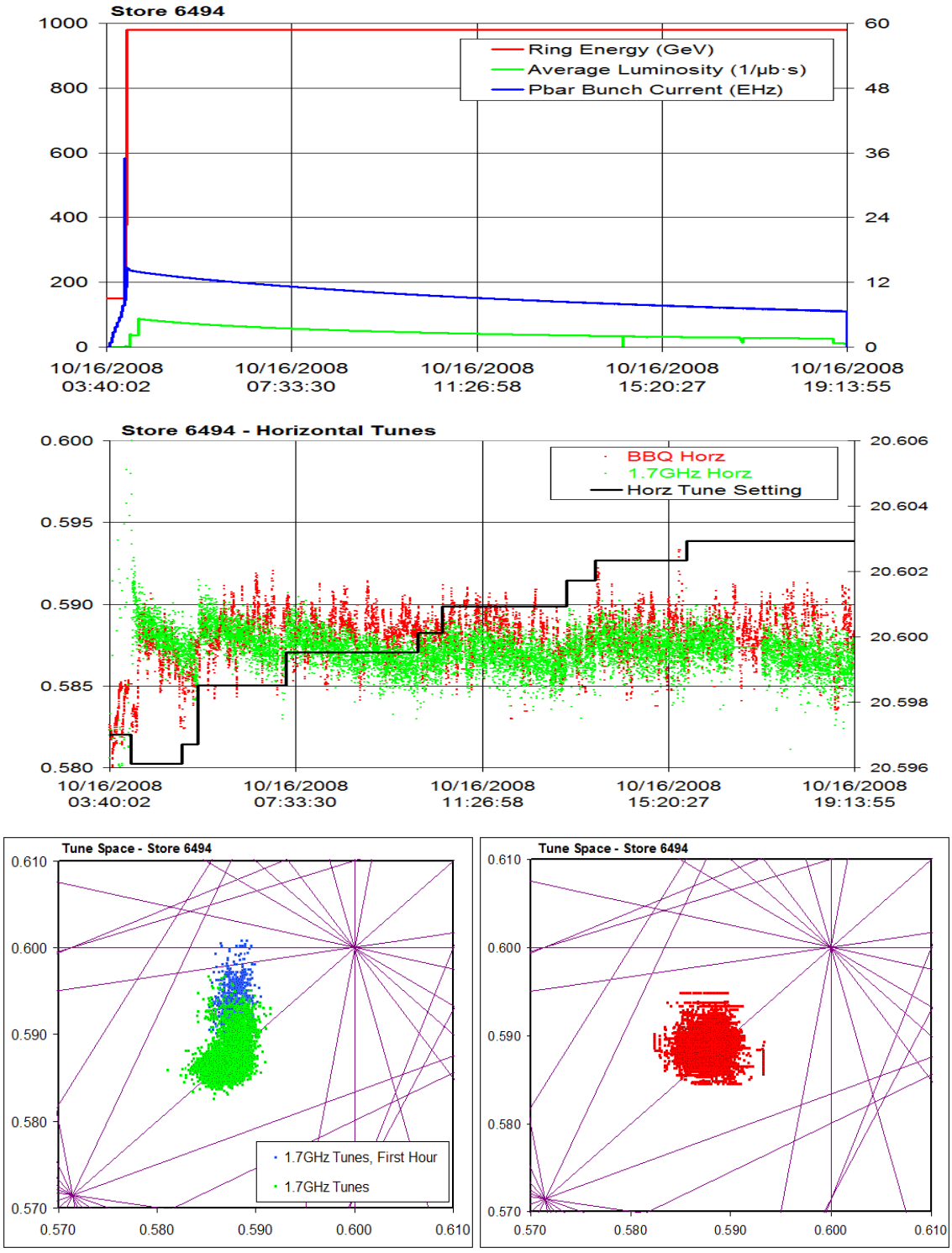

Figure 52 – A summary for Store 6494. This was during a period of smaller stores due to an instability. The store summary is located at the top. The horizontal antiproton tunes were adjusted as needed as shown in the middle plot. Tune space diagrams of 1.7GHz Schottky tunes and BBQ tunes are shown at the bottom left and right respectively.

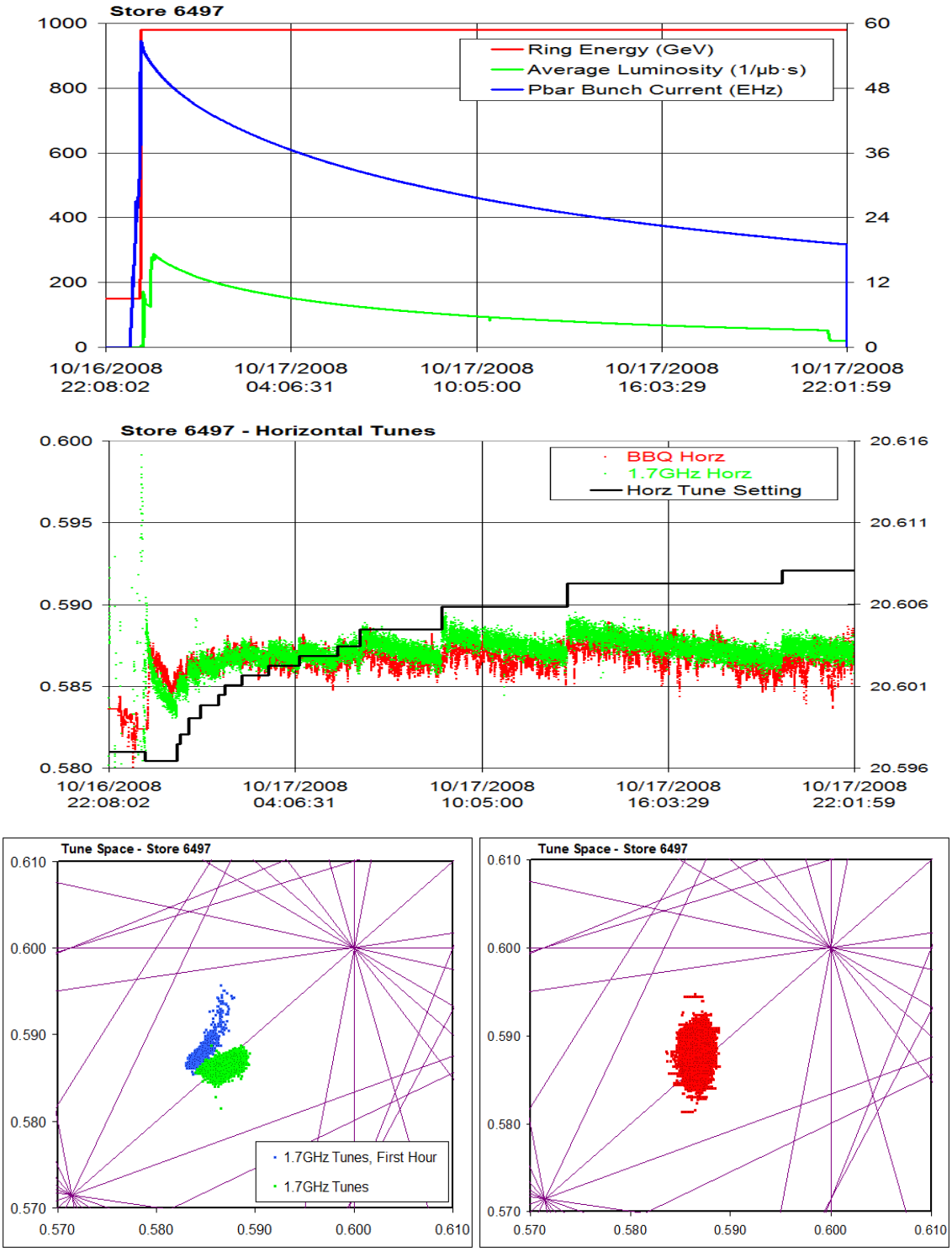

Figure 53 – A summary for Store 6497. This was during a period of smaller stores due to an instability. The store summary is located at the top. The horizontal antiproton tunes were adjusted as needed as shown in the middle plot. Tune space diagrams of 1.7GHz Schottky tunes and BBQ tunes are shown at the bottom left and right respectively.

The BBQ antiproton tune measurements were noisier than the proton tune measurements seen in stores 6539, 6540 and 6541 above. They were also noisier than the antiproton measurements from the 1.7GHz Schottky, particularly in the vertical plane, but the BBQ tunes remained more localized to a believable tune space. As with the proton stores, the antiproton 1.7GHz Schottky tunes were spread near the $^{3}/_{5}^{ths}$ resonance tune line of both planes, which would have resulted in catastrophic beam loss if the tune measurements were believable. The BBQ signal is similar to the proton fit to Store 5590 shown in *Figure 35*.

Store 6492, summarized in *Figure 51*, was established with a smaller number of antiprotons than usual due to instabilities that were being investigated at the time. The BBQ tune follows the 1.7GHz Schottky loosely, but appears to frequently lose lock on a believable tune as indicated by the numerous vertical bands scattered about the 1.7 GHz Schottky measurement.

Store 6494, summarized in *Figure 52*, had a very low bunch current effectively acting as a continuation of store 6492. This was primarily due to problems with antiproton transfers into the Tevatron, which led to an even more severely limited quantity of antiprotons for use in the Store. The tune measurement on this store proved difficult even for the 1.7GHz Schottky, which even stopped reporting back tune measurements for a stretch later in the store presumably due to a poor fit.

Store 6497, summarized in *Figure 53*, had the highest initial bunch current and, as a result, the best initial fit. The Tune-Space plot of the BBQ data still appears quite tall indicating a large spread in the vertical plane as can be seen in *Figure 54*. The horizontal BBQ tunes appear to be on par with the 1.7GHz Schottky near the beginning of the store, when the bunch current is greatest, but it begins to wander later in the store as the bunch current decreases.

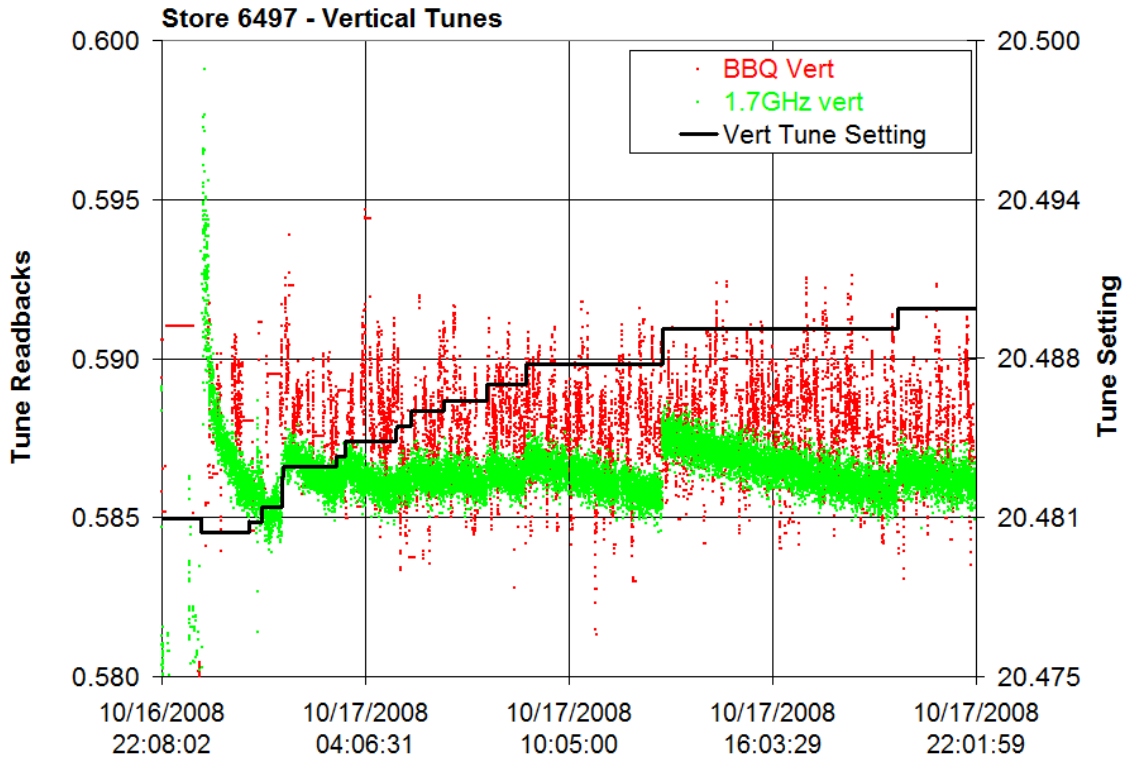

Figure 54 – Vertical tune data from Store 6497. This complements the Horizontal data in *Figure 53* to produce the tune-space plot in the same figure.

Recall from the off-line analysis of store 6413 that a threshold bunch current, J_{thr}, of 39EHz rendered even the horizontal tune measurement useless. The data from store 6497 shows considerable improvement, lowering the threshold bunch current below 20EHz in the Horizontal plane. The vertical plane, however, still requires improvement since reliable tune measurement is still not present at the initial bunch current of about 56EHz.

Conclusions

Currently, the BBQ tune measurement system is being used to measure the proton tunes in the Tevatron. These appear to be as reliable if not more reliable than the 1.7GHz tune measurement system, which is still used as the primary tune measurement system on the Tevatron at Fermilab. In its role as a back-up tune monitoring system it has been employed in analysis of Tevatron tune problems and is ready to stand in as the primary proton tune measurement system in the event that the 1.7GHz system should malfunction.

Measurement of the antiproton tunes in the Tevatron is still somewhat limited and will require more work to adequately provide measurements of the antiproton tunes from the BBQ tune measurement system. Among the challenges is to lower the threshold bunch current, J_{thr}, to allow for more reliable tune measurements with fewer antiprotons in the Tevatron by continuing to work with attenuation and filtering of the signals prior to the diode detectors and the BBQ front end.

One limitation of the BBQ tune measurement system at this time is that there is currently only one VSA available for the project and hence measurements can only be made for two planes. The BBQ is currently hooked up to the proton vertical and horizontal pickups. Another BBQ front end, however, is available for measurement of the Antiproton tunes. As an alternative to a VSA, Marek Gasior has noted that the signals from the VSA front end are in the audible range, or between 20Hz and 20 kHz, which could be recorded using a standard sound card. In principle, applying a Fourier transform to the recorded track will result in a comparable spectrum to that of a VSA. A local computer could record sound files, perform the Fourier transform on the data, extrapolate the tune in the same manner as PA4040 and provide the tunes for use in ACNET.

Appendix A – pa4040.cpp

```
/*
**  Copyright 2001, Universities Research Association.  All rights reserved.
**
**  ABSTRACT:   This program PA4040 is for TeV tune analysis from the BBq VSA
**      at E0.
**
**  AUTHORS:    Chip Edstrom
**
**  CREATION DATE:     14-MAR-08
**
**  LAST LINT DATE:
**
**  LINT EXCEPTIONS:
**
**  MODIFICATION HISTORY: See history.txt at
**  http://www-ad.fnal.gov/cgi-cvs/cvsweb.cgi/mecca/pas/pa4040/
**
*****************************************************************************/

using namespace std;

/*****************************************************************************
*
*       Include Files
*
*****************************************************************************/

#include "acnet_errors.h"       // Defs for ACNET errors
#include "cbslib.h"             // CBS const and prototypes
#include "cns_data_structs.h"   // Console data structures
#include "cnsparam.h"           // Generic console constants
#include "clib.h"               // Old CLIB const and prot
#include "diolib.h"             // DIO const and prototypes
#include "dbprops.h"            // Database properties
#include "extchrset.h"          // Extended character set
#include "argument_defs.h"      // General argument definitions

#include <math.h>
#include <time.h>

#include <iostream>             //
#include <fstream>              // stream prototypes
#include <sstream>              //
#include <string>

// VSA IP prototypes
#include "stdio.h"              // for fprintf and NULL
#include "stdlib.h"             // for 'ctof()'
#include "string.h"             // for memcpy and memset
#include "errno.h"              // for strerror
#include "sys/socket.h"         // for connect and socket
#include "netinet/in.h"         // for sockaddr in
#include "netdb.h"              // for gethostbyname

// External prototypes
extern "C" {
    #include "lm.h"             // /usr/local/mecca_head/mecca/uls/ul_lma/
}

/*****************************************************************************
*
*       Global defs
*
*****************************************************************************/
// General Program Options
#define TITLE           "Chip's BBq VSA Program"
#define LOG_FILE_NAME   "PA4040LG"
```

```
#define ERROR_LOG_FILE  "PA4040ER"
#define BKG             WMNGR BACKGROUND
#define DEBUG           false           // Allows for debugging options
#define SCREEN ROWS     10              // Number of rows in background window
#define SCREEN_COLUMNS  40              // Number of columns in background window
#define PGM_ERR         true            // Initialize the Error Window and/or log errors
& messages
#define PGM UTIL        false           // Initialize the Program Utilities?
#define MB_ITEMS        1               // Items in the menu bar
#define MB ROW          3               // Row in which to place the menu bar
#define NROWS_ERR       4               // Number of rows allocated for the message
window
#define ROW_ERR         (SCREEN_ROWS - NROWS_ERR - (2 * WMNGR_BORDER_WIDTH) + 1)
#define COL_ERR         2               // width of the message window frame
#define LEN_ERR         (SCREEN_COLUMNS - (2 * WMNGR_BORDER_WIDTH))
#define COL UTIL        (SCREEN COLUMNS - 10)
#define ROW_UTIL        4
#define FATAL           true            // Constant for a fatal error
#define NONFATAL        false           // Constant for a non-fatal error (default)
#define TAB             9               // The ASCII character code for Tab

// Measurement Toggle Constants
#define T_FAIL          -1
#define T_OFF           0
#define T_ON            1

// Tune VSA constants
#define SCOPENAME       "131.225.128.6" // The name (*.fnal.gov) or IP address of the
scope
#define SCOPESOCKET     5025            // Socket used to contact VSA
#define SCOPEDELAY      2               // Number of seconds between measurements
#define AVEQ            0.02            // Averaging quotient (relative weight applied to
new data)
#define h               1113            // The harmonic number of TeV
#define REV FREQ        47713.11        // TeV Revolution Frequency
#define PRECISION       4               // Precision of the values reported by VSA
#define POINTSIZE       (PRECISION + 8) // SizeOf single VSA point
#define HORZ            1               // Default horz fit parameters
#define VERT            2               // Default vert fit parameters

// Spike-stripping constants
#define SPIKESTRIP      true            // Strip the spikes?
#define THRESH_SS       2               // Spike Threshold

// LMA constants
#define USE_JACOBIAN    true            // USE THE JACOBIAN - Will probably result in a
quicker fit
#define NPARAMS         5               // NUMBER OF PARAMETERS TO FIT
#define MAXITER         100             // Maximum number of LMA iterations before giving
up.

// Parameter Feed-Forward constants
#define THRESH_FF       1000            // Threshold for feeding forward the fit
parameters as based on finfo[]

// ACNET Parameters for fit parameter output
#define N               NPARAMS+3       // Total Number of Parameters (fitting or
otherwise)
#define T_BBQVSA        223676          // VSA interface parameter

#define T_BBQPHA        223625          // Amplitude DI
#define T_BBQPHQ        223624          // Tune DI
#define T_BBQPHS        223622          // Sigma DI
#define T BBQPHM        223626          // Linear Slope DI
#define T_BBQPHL        223627          // Linear Offset DI
#define T_BBQPHE        223632          // Horz fit error DI (from info[1])
#define T_BBQPHN        223635          // Number of points used for Horz fit
#define T_PHQAVG        226594          // Averaged horizontal tune parameter DI

#define T BBQPVA        223628          // Amplitude DI
#define T_BBQPVQ        223629          // Tune DI
```

```
#define T_BBQPVS        223623          // Sigma DI
#define T_BBQPVM        223630          // Linear Slope DI
#define T_BBQPVL        223631          // Linear Offset DI
#define T_BBQPVE        223633          // Vert fit error DI (from info[1])
#define T_BBQPVN        223636          // Number of points used for Vert fit
#define T_PVQAVG        226595          // Averaged vertical tune parameter DI

// Plot file output
#define OUTPUT          true            // Save the data to file for plotting
#define FILENAME        "qfile"         // The output file for data
#define WORKDIR         "/usr/local/cbs_files/cns_write/tevatron"

/*****************************************************************************
*
*       Module variables & structs
*
*****************************************************************************/
bool    fatal_error = false;
bool    gui_interface;

struct  lockstruct  { LOCK_ENTRY_DATA data; char name[LOCK_MAX_NAME_LEN]; bool set; int
status; };
struct  vsastate    { bool setup; bool measure; bool busy; int overrun; double cent;
double span; int npoints; int tbs; };
struct  scopecomm   { int socket; FILE* file; int inst;};
struct  dataset     { double* q; double* x; double* info; int n; char* buffer; };
struct  paramset    { double* set; float* nom; float* tol; int* di; int chan; };
struct  requests    { bool poll; bool kill; bool reset; bool rescale; bool feedforward;
bool returntozero; bool toggle; bool abort; };
struct  tm * timeinfo;

/*****************************************************************************
*
*       Module internal routines
*
*****************************************************************************/
static void pgm_ini(lockstruct& lock);
static void pgm_trm(lockstruct& lock);
static void pgm_per(requests& req);
static void pgm_kbd(short wid, int row, int col, requests& req);
static void pgm_err(int err_code, char* text, int color = RED, bool fatal = false);
static void pgm_msg(char* text, int color = GREEN, int log = ERR_NO_LOG);

bool        scopeInit(vsastate& vsa, scopecomm& scope);
int         scopeSpace(vsastate& vsa, scopecomm& scope);
char*       scopeCmd(const char *command, char *result, size_t maxLength, vsastate& vsa,
scopecomm& scope);

int         tuneToggle(vsastate& vsa, scopecomm& scope, lockstruct& lock, paramset& Px,
paramset& Py, requests& req);
void        tuneCoord(vsastate& vsa, scopecomm& scope, dataset data, paramset& P,
requests& req);
void        tuneReset(paramset& P, requests& req);
void        tuneSet(paramset& P);
void        tuneGetNT(paramset& P);
void        tuneWrite(double* q, double* x, int n, int m);

double      f(double q, double* p);
int         lma(double* x, double* y, int npoints, double* p, double* info);
extern "C" {
    void    ftest(double* p, double* y, int nparams, int npoints, void* xx);
    void    jftest(double*p, double *dyda, int nparams, int npoints, void*xx);
}

void        lockSet(lockstruct& lock);
void        lockUnset(lockstruct& lock);
```

```
/*****************************************************************************
*
*     status.i4.v = main()
*
*       Main program - get interrupt (event) type and call handler routine
*
*****************************************************************************/
int main(void) {
    char        msg[ERR_MAXLEN];
    int         column;
    int         info;
    int         row;
    short       int_type;
    short       window_id;
    bool        color = false;          // Color flip-flop variable
    requests    req;                    // Requests for scope interaction
    int         t_bbqvsa = T_BBQVSA;    // di for T:BBQVSA for initial set
    time_t      ta = time(NULL);        // time of previous measurement
    time_t      tb = time(NULL);        // current time for comparison to ta;
    vsastate    vsa;                    // VSA setup, measurement, busy states…
    scopecomm   scope;                  // Scope Communications
    lockstruct  lock;                   // lock info

    // Program initialization
    lock.set = false;       vsa.setup = false;          vsa.busy = false;
    vsa.npoints = 0;        vsa.measure = false;        pgm_ini(lock);
    req.poll = true;        req.kill = false;           req.reset = false;
    req.rescale = false;    req.feedforward = true;     req.returntozero = true;
    req.abort = false;

    timeinfo = localtime ( &tb );
    pgm_msg("-------------------------------------------------------",GREEN,ERR_LOG_IT);
    sprintf(msg,"Console: CLX%i",myconsole()); pgm_msg(msg,GREEN,ERR_LOG_IT);
    sprintf(msg,"Time: %s", asctime(timeinfo)); pgm_msg(msg,GREEN,ERR_LOG_IT);

    pgm_err( dio_on(&t_bbqvsa) ,"T:BBQVSA control error");

    if ( lock.set ) { scopeInit(vsa, scope); }
    if ( !vsa.setup ) { pgm_msg( "Unable to setup scope" ,RED); }

    vsa.tbs = (POINTSIZE*vsa.npoints) + 1;  // Total buffer size required for scope data
    char        dataBuf[vsa.tbs];           // Data buffer for scope data call

    double      finfo[LM_INFO_SZ];          // Fit info
    double      q[vsa.npoints];             // Raw Data Tunes
    double      x[vsa.npoints];             // Raw Data

    // All parameter indices are from 0 to NPARAMS+1 (the last two, NPARAMS and
    // NPARAMS+1) are reserved for the error and npoints used for the fit. 0 to
    // NPARAMS-1 are used for the actual fitting parameters.
    double      xP[N];                  // Horz fit parameters
    double      yP[N];                  // Vert fit parameters
    float       xPn[N];                 // Nominal horz fit parameters
    float       yPn[N];                 // Nominal vert fit parameters
    float       xPt[N];                 // Tolerance of horz fit parameters
    float       yPt[N];                 // Tolerance of vert fit parameters
    static int  xdi[N];                 // Device Indices for each of the Horz parameters
    static int  ydi[N];                 // Device Indices for each of the Vert parameters

    dataset  data;          data.q = q;             data.x = x;
    data.info = finfo;      data.n = vsa.npoints;   data.buffer = dataBuf;
    paramset Px;    Px.set = xP; Px.nom = xPn; Px.tol = xPt; Px.di = xdi; Px.chan = HORZ;
    paramset Py;    Py.set = yP; Py.nom = yPn; Py.tol = yPt; Py.di = ydi; Py.chan = VERT;

    // Set parameter DIs
    Px.di[0] = T_BBQPHA; Px.di[1] = T_BBQPHQ; Px.di[2] = T_BBQPHS; Px.di[3] = T_BBQPHM;
    Px.di[4] = T_BBQPHL; Px.di[5] = T_BBQPHE; Px.di[6] = T_BBQPHN; Px.di[7] = T_PHQAVG;

    Py.di[0] = T_BBQPVA; Py.di[1] = T_BBQPVQ; Py.di[2] = T_BBQPVS; Py.di[3] = T_BBQPVM;
    Py.di[4] = T_BBQPVL; Py.di[5] = T_BBQPVE; Py.di[6] = T_BBQPVN; Py.di[7] = T_PVQAVG;
```

```
    while(TRUE){
        if ( fatal_error ) return -1;
        window_intype(&window_id,&int_type,&row,&column,&info);

        switch(int_type){
            case INTINI:            // Initialization interrupt (event)
                Px.set[7] = 0; Py.set[7] = 0; // set averaged parameters to 0 at start
                break;
            case INTTRM:            // Termination interrupt (event)
                timeinfo = localtime ( &tb );
                pgm_trm(lock);
                sprintf(msg,"Program Terminated: %s", asctime(timeinfo));
pgm_msg(msg,GREEN,ERR_LOG_IT);
                pgm_msg("-----------------------------------------------------
",GREEN,ERR_LOG_IT); pgm_msg("",GREEN,ERR_LOG_IT);
                break;
            case INTKBD:            // Keyboard/user interrupt (event)
                pgm_kbd(window_id, row, column, req);
                break;
            case INTPER:            // Periodic interrupt (event) (~15Hz)
                if ( lock.set && vsa.setup ) pgm_per(req);
                break;
            default:                // Other interrupts (events)
                break;
        }

        // External parameter control
        if ( req.kill ) return 0;

        if ( req.toggle != vsa.measure ) {
            if ( tuneToggle(vsa, scope, lock, Px, Py, req) == T_ON ) {
                Px.set[7] = 0; Py.set[7] = 0; // set averaged parameters to 0 at start
                pgm_err( dio_on(&t_bbqvsa) ,"T:BBQVSA control error");
            } else {
                pgm_err( dio_off(&t_bbqvsa) ,"T:BBQVSA control error");
            }
        }

        if ( req.reset ) {
            req.reset = false;
            pgm_msg("T:BBQVSA parameter reset request",GREEN,ERR_LOG_IT);
            tuneReset(Px, req); tuneReset(Py, req);
        }

        if ( req.rescale ) {
            req.rescale = false;
            scopeCmd("disp:wind1:trac:y:auto once", data.buffer, vsa.tbs, vsa, scope);
            scopeCmd("disp:wind2:trac:y:auto once", data.buffer, vsa.tbs, vsa, scope);
        }

        if ( req.abort ) {
            req.abort = false;
            scopeCmd("abor", data.buffer, vsa.tbs, vsa, scope);
        }

        // Perform measurements every SCOPEDELAY seconds
        if ( (tb = time (NULL)) - ta >= SCOPEDELAY  && vsa.setup ) {
            time(&ta);
            if ( !vsa.busy && vsa.measure ) {
                sprintf(msg, "Still polling %s...",SCOPENAME);
                if ( color ) {
                    pgm_msg(msg, CYAN); color = false;
                } else {
                    pgm_msg(msg, BLUE); color = true;
                }
                tuneCoord(vsa, scope, data, Px, req);
                tuneCoord(vsa, scope, data, Py, req);
            } else if( vsa.busy ) { } // this means I'm running over myself
            req.poll = true;
```

```
        }
    }
} //end main()

/*****************************************************************************
*
*     pgm_ini()
*
*     Initialize program (This is called once after the program is loaded.)
*
*****************************************************************************/
static void pgm_ini(lockstruct& lock) {
        static char *menu_text[] = {        // menu bar text items
            "<< Toggle Measurement On/Off >>"};
    char        *menu_text_ptr;

    if (myslot() == NONUSER_SLOT) {
        gui_interface = false;
    } else {
        gui_interface = true;
    }

    if ( gui_interface ) {
        //INITIALIZE WINDOW AND DISPLAY TITLE
        window_set_background_size_c(SCREEN_ROWS,SCREEN_COLUMNS);
        window_restore_hint_c(WMNGR_REPAINT);
        window_enable_interrupts();

        //Print the title
        window_center_text_c(BKG,1,TITLE,0,CYAN);

            //Create error window at bottom of page, enable error reporting.
        if (PGM_ERR) error_init_c(
                ROW_ERR,COL_ERR,LEN_ERR,LOG_BOTH,LOG_FILE_NAME,
                NO_AUTHOR,NROWS_ERR,USE_WINDOW_MANAGER,
                ERR_DEFAULT_PRIORITY,WMNGR_BORDER_THIN,
                ERR_NO_ERASE,ENABLE_ERROR_TEXT);

        // Install Program tools in upper right corner
        if (PGM_UTIL) utility_window_init_c(ROW_UTIL,COL_UTIL);

        //create the menu bar
        build_menu_bar_text(menu_text,&menu_text_ptr,MB_ITEMS);
        menu_bar_create(menu_text_ptr,NULL,MB_ROW);

    } else {
        if (PGM_ERR) error_init_c(-1,1,90,LOG_SHR,LOG_FILE_NAME,NO_AUTHOR);
    }

    // enable file management
    fm_tuner();

    // set program lock
    sprintf(lock.name, "%s_LOCK", SCOPENAME);
    lockSet(lock);

} //end pgm_ini
```

```
/*****************************************************************************
*
*     pgm kbd(win id.i2.v,row.i4.v,col.i4.v)
*
*     Decode keyboard/user interrupts (events)
*
*****************************************************************************/
static void pgm_kbd(short wid, int row, int col, requests& req) {
    int item_bar;

    item_bar = menu_bar_update();
    switch (item_bar) {
        case 1:
            if (req.toggle) { req.toggle = false; } else { req.toggle = true; }
            break;

        default:
            break;
    }
    return;
} //end kbd_int

/*****************************************************************************
*
*     pgm_per()
*
*     Decode periodic interrupts (events)  and look for interface cues from T:BBQVSA(~15
Hz)
*
*****************************************************************************/
static void pgm_per(requests& req) {
    int     di = T_BBQVSA;
    int on_status;
    float   value;

    if ( req.poll ) {
        req.poll = false;

        pgm_err( dio_get_dev_c(di,PRSET,&value) ,"T:BBQVSA aPoll error");
        if ( value != 0 ) {
            if ( value == -1 ) {
                req.kill = true;
            } else if( value == 1 ) {
                req.rescale = true;
            } else if( value == 2 ) {
                req.feedforward = false;
            } else if( value == 3 ) {
                req.feedforward = true;
            } else if( value == 4 ) {
                req.returntozero = false;
            } else if( value == 5 ) {
                req.returntozero = true;
            } else if( value == 6 ) {
                req.abort = true;
            } else {
                req.reset = true;
            }
            value = 0;
            pgm_err( dio_set_dev_c(di,&value) ,"T:BBQVSA aSet error");
        }

        pgm_err( dio_is_on_c(di,&on_status) ,"T:BBQVSA dPoll error");
        if ( on_status == TRUE ) { req.toggle = true; } else { req.toggle = false; }
    }
}   //end pgm_per
```

```
/*****************************************************************************
*
*     pgm_trm(void)
*
*     Program termination (This is called once as the program exits.)
*
*****************************************************************************/
static void pgm_trm(lockstruct& lock) {
    if ( lock.set ) {
        lockUnset(lock);            // release the lock on the VSA
    }
    return;
}   //end pgm_trm

/*****************************************************************************
*
*   pgm_err(text.i1a.r, color.i4.v, fatal.l4.v [,error_code.i4.v])
*
*   Prints an error message to the error window in a specified color and
*   terminates the program if it is fatal.
*
*****************************************************************************/
static void pgm_err(int err_code, char *text, int color, bool fatal) {
    char    msg[ERR_MAXLEN];

    if ( err_code == OK ) return; // Nothing's wrong = nothing to report... Return.

    if ( PGM_ERR ) {

        // ACNET Error Facility & Error Code from acnet_errors.h
        int fac = int( fmod(err_code,256) );
        if ( err_code < 0 ) fac = 256 + fac;
        int err = int((err_code - fac)/256);

        ACNET_ERR       acnet_error = {fac,err};
        char            err_name[ACNET_ERROR_TEXT_LEN];
        char            err_text[ERR_MAXLEN];
        int             status = acnet_error_text(&acnet_error,err_name);

        if ( status != OK ) error_message_c("Error getting error info (how
ironic)",ERR_SIMPLE_DISPLAY,RED,ERR_LOG_IT);

        sprintf(err_text,"%s::%s",err_name,text);
        error_message_c(err_text,ERR_SIMPLE_DISPLAY,color,ERR_LOG_IT);

        if ( DEBUG ) { sprintf(msg,"(Error code: %i %i [%i])",fac,err,err_code);
pgm_msg(msg, MAGENTA); }
    }

    if ( fatal ) fatal_error = true;
    return;

} //end of pgm_err()

/*****************************************************************************
*
*   pgm msg(text.i1a.r, color.i4.v, [log.i4.v])
*
*   Prints a message to the error window in a specified color.
*
*****************************************************************************/
static void pgm_msg(char *text, int color, int log) {

    if ( DEBUG ) color = BLUE;
    if ( PGM_ERR ) error_message_c(text,ERR_SIMPLE_DISPLAY,color,log);
    return;

} //end of pgm_msg()
```

```
/*****************************************************************************
*
*     scopeCmd
*
*     Sends a command to the VSA
*
*****************************************************************************/
char* scopeCmd(const char *command, char *result, size_t maxLength, vsastate& vsa,
scopecomm& scope) {
    if ( DEBUG ) pgm_msg("scopeCmd()");
    size_t  length;
    char    msg[ERR_MAXLEN];

    if ( DEBUG ) { sprintf(msg, "Command to %s : %s", SCOPENAME, command); pgm_msg(msg,
GREEN); }

    if ( vsa.setup ) {

        if (fprintf(scope.file,"%s\n", command) < 0) return NULL;
        fflush(scope.file);
        if (!strchr(command,'?')) return NULL;
        if (fgets(result, maxLength, scope.file) == NULL) return NULL;
        length = strlen(result);
        if (result[length-1] == '\n') result[length-1] = ',';

        if ( DEBUG ) {
            if (length < 30)       { sprintf(msg, "%s returns : %s", SCOPENAME, result);
pgm_msg(msg);
                    } else { sprintf(msg, "%s returns : A very long string", SCOPENAME);
pgm_msg(msg);}}

        return result;
    } else { return NULL; }
}

/*****************************************************************************
*
*     scopeInit()
*
*     Initializes the scope socket
*
*****************************************************************************/
bool scopeInit(vsastate& vsa, scopecomm& scope) { if ( DEBUG ) pgm_msg("scopeInit()");
    struct      hostent     *hostPtr;
    struct      sockaddr_in     peeraddr_in;
    char        msg[ERR_MAXLEN];

    if ( vsa.setup ) return vsa.setup; // already setup... nothing to do.

    pgm_msg("Initializing Scope");
    hostPtr = gethostbyname(SCOPENAME);
    if ( hostPtr == NULL ) {
        sprintf(msg, "Unable to resolve hostname: %s", SCOPENAME);
        pgm_err(NETAPI_BAD_HOSTNAME,msg);
        return vsa.setup;
    }

    scope.socket = socket(AF_INET, SOCK_STREAM, 0); // AF_INET = 2, SOCK_STREAM = 1
    if ( scope.socket == -1 ) {
        sprintf(msg, "%s", strerror(errno));
        pgm_err(NETAPI_SOCKET_FAILURE,msg);
        return vsa.setup;
    }

    memset( &peeraddr_in , 0, sizeof(struct sockaddr_in));
    memcpy( &peeraddr_in.sin_addr.s_addr , *(hostPtr->h_addr_list), sizeof(struct
sockaddr_in));
    peeraddr_in.sin_family = AF_INET; peeraddr_in.sin_port = htons(SCOPESOCKET);
    if ( connect(scope.socket, (struct sockaddr *) &peeraddr_in, sizeof(struct
sockaddr_in)) == -1 ) {
```

```
        sprintf(msg, "%s", strerror(errno));
        pgm_err(NETAPI_BIND_FAILURE,msg);
        return vsa.setup;
    }

    if ( scope.socket == -1 ) {
        sprintf(msg,"Null socket returned");
        pgm_err(NETAPI_SOCKET_FAILURE,msg);
        return vsa.setup;
    }

    scope.file = fdopen(scope.socket,"r+");
    if (scope.file == NULL) {
        sprintf(msg, "%s", strerror(errno));
        pgm_err(NETAPI_IOCTL_FAILURE, msg);
        return vsa.setup;
    }

    sprintf(msg, "%s Initialized", SCOPENAME); pgm_msg(msg,GREEN,ERR_LOG_IT);
    vsa.setup = true;
    scopeSpace(vsa, scope); // Get initial conditions
    return vsa.setup;
}

/*****************************************************************************
*
*     scopeSpace()
*
*     Sets the precision for readbacks and gets the Span, Center Frequency
*     and number of points across the sample for use in the program.
*
*****************************************************************************/
int scopeSpace(vsastate& vsa, scopecomm& scope) { if ( DEBUG ) pgm_msg("scopeSpace()");
    char    s_cmd[256];
    char    s_msg[256];

    sprintf(s_cmd, "form:data ascii,%i", PRECISION);
    scopeCmd(s_cmd, s_msg, sizeof(s_msg), vsa, scope);

    scopeCmd("calc:data:head:poin?", s_msg, sizeof(s_msg), vsa, scope);
    if ( s_msg != NULL ) {
        vsa.npoints = atoi(s_msg);
    } else {
        return -1;
    }

    scopeCmd("freq:cent?", s_msg, sizeof(s_msg), vsa, scope);
    if ( s_msg != NULL ) {
        vsa.cent = atof( s_msg );
    } else {
        return -1;
    }

    scopeCmd("freq:span?", s_msg, sizeof(s_msg), vsa, scope);
    if ( s_msg != NULL ) {
        vsa.span = atof( s_msg );
    } else {
        return -1;
    }
    return 0;
}
```

```
/*****************************************************************************
*
*     tuneCoord()
*
*   Tune Measurement coordinator.
*
*****************************************************************************/
void tuneCoord(vsastate& vsa, scopecomm& scope, dataset data, paramset& P, requests& req)
{
    if ( DEBUG ) pgm_msg("tuneCoord()");
    char    s_msg[256];
    char    point[POINTSIZE];
    int i, j;
    bool    ff_outoftol = false;

    vsa.busy = true; {

        // Refresh the tune points for this round

        for ( i = 0; i < vsa.npoints; i++ ) {
            data.q[i] = 1 - ((vsa.cent - (.5 * vsa.span)) + (i * vsa.span/ vsa.npoints))
/ REV_FREQ;
        }

        // Get data from the appropriate channel and split into arrays data.x[0..n]
        sprintf(s_msg,"calc%i:data?", P.chan);
        scopeCmd(s_msg, data.buffer, vsa.tbs, vsa, scope);
        for ( i = 0; i < vsa.npoints; i++ ) {
            for ( j = 0; j < POINTSIZE; j++ ) { point[j] = data.buffer[(i*POINTSIZE) +
j]; }
            data.x[i] = atof(point);
        }

        // Output the data to file for plotting
        if ( OUTPUT ) tuneWrite(data.q, data.x, vsa.npoints, P.chan);

        // Perform an initial LMA fit
        lma(data.q,data.x,vsa.npoints,P.set,data.info);

        if ( SPIKESTRIP ) {
            // Strip values over THRESH_SS above the initial fit
            for ( j = 0, i = 0; i < vsa.npoints; i++ ) {
                if ( data.x[i] < f(data.q[i], P.set) + THRESH_SS) {
                    data.q[j] = data.q[i];
                    data.x[j] = data.x[i];
                    j++;
                }
            }

            // Perform one more fit on stripped data
            lma(data.q,data.x,j,P.set,data.info);
        }

        // Output the fit parameters to ACNET parameters for datalogging and feedforward
        P.set[5] = data.info[1]; P.set[6] = float(j); P.set[2] = fabs(P.set[2]);

        if ( P.set[7] != 0 ) { P.set[7] = ((1-
float(AVEQ))*P.set[7])+(float(AVEQ)*P.set[1]); } else { P.set[7] = P.set[1]; }

        for ( i=0; i<NPARAMS+2;i++ ) {
            if ( P.set[i] > P.nom[i] + fabs(P.tol[i]) || P.set[i] < P.nom[i] -
fabs(P.tol[i])) {
                ff_outoftol = true;
            }
        }

        if ( ff_outoftol || (!req.feedforward) ) {
            if ( DEBUG ) pgm_msg("Resetting - No Feed Forward");
            tuneReset(P, req);
        } else {
```

```
            if ( DEBUG ) pgm_msg("Feeding-forward");
            tuneSet(P);
        }

    } vsa.busy = false;
}

/*****************************************************************************
*
*     f()
*
*     This description is longer than the function it describes. It returns
*     the value of the fit for any tune point with a supplied set of of fit
*     parameters.
*
*****************************************************************************/
double f(double q, double* p) { return (p[0]*exp(-0.5*((double) pow((q-
p[1])/p[2],2)))+(p[3]*q)+p[4]); }

/*****************************************************************************
*
*     tuneToggle()
*
*   Toggle the measurement off and on as well as establish defaults
*   for the beginning of measurement (via scopeSpace & tuneGetNT)
*
*****************************************************************************/
int tuneToggle(vsastate& vsa, scopecomm& scope, lockstruct& lock, paramset& Px, paramset&
Py, requests& req) {
    if ( DEBUG ) pgm_msg("tuneToggle()");
    char    msg[ERR_MAXLEN];

    if ( vsa.setup ) {
        if ( lock.set ) {
            if ( vsa.measure == false ) {
                if ( scopeSpace(vsa, scope) != -1 ) {
                    pgm_msg("Tune Measurement on...",GREEN,ERR_LOG_IT);
                    tuneReset(Px, req); tuneReset(Py, req);
                    vsa.measure = true;
                    return T_ON;
                } else {
                    pgm_err(NETAPI_RECV_ERROR, "Unable get scope-space parameters...
aborting.");
                    return T_FAIL;
                }
            } else {
                pgm_msg("Tune Measurement off...",GREEN,ERR_LOG_IT);
                vsa.measure = false;
                return T_OFF;
            }
        } else {
            sprintf(msg,"Lock not established");
            pgm_err(lock.status,msg);
            return T_FAIL;
        }
    } else {
        sprintf(msg, "%s not setup! (Rerun pa4040)", lock.name);
        pgm_msg(msg, YELLOW);
        return T_FAIL;
    }
}
```

```
/*****************************************************************************
*
*     tuneReset()
*
*     Outputs fit parameters to ACNET parameters
*
*****************************************************************************/
void tuneReset(paramset& P, requests& req) { if ( DEBUG ) pgm_msg("tuneReset()");
    int     i;

    tuneGetNT(P);
    if ( req.returntozero ) {
        for ( i=0;i<NPARAMS;i++ ) {
            P.set[i] = 0;
        }
        tuneSet(P);
    }
    for ( i=0;i<NPARAMS;i++ ) {
        P.set[i] = double(P.nom[i]); // The most frustrating line EVAR!
    }
}

/*****************************************************************************
*
*     tuneSet()
*
*     Outputs fit parameters to ACNET parameters
*
*****************************************************************************/
void tuneSet(paramset& P) { if ( DEBUG ) pgm_msg("tuneSet()");
    char    msg[ERR_MAXLEN];
    int i;
    float   Pf;

    for ( i=0;i<N;i++ ) {
        Pf = float(P.set[i]);
        sprintf(msg,"Error setting %i",P.di[i]);
        pgm_err( dio_set_dev_c(P.di[i],&Pf) ,msg);
    }
}

/*****************************************************************************
*
*     tuneGetNT()
*
*     Gets the nominal and tolerances for the parameters
*
*****************************************************************************/
void tuneGetNT(paramset& P) { if ( DEBUG ) pgm_msg("tuneGetNT()");
    char    msg[ERR_MAXLEN];
    int     i;

    for ( i=0;i<N;i++ ) {
        sprintf(msg,"%i alarm limits",P.di[i]);
        pgm_err( dio_alarm_limits_c(P.di[i],&P.nom[i],&P.tol[i]) ,msg);
    }
}

/*****************************************************************************
*
*     tuneWrite()
*
*     Outputs data to file
*
*****************************************************************************/
void tuneWrite(double* q, double* x, int n, int m) { if ( DEBUG ) pgm_msg("tuneWrite()");
    char        msg[ERR_MAXLEN];
    char        filename[10];
    char        outfile[256];
    ofstream    fout;
```

```
    int         i;

    if (m == HORZ) { sprintf(filename,"%s%s",FILENAME,"h"); }
    else { sprintf(filename,"%s%s",FILENAME,"v"); }

    if ( DEBUG ) { sprintf(msg,"Opening %s for output...",filename); pgm_msg(msg); }

    sprintf(outfile, "%s/%s", WORKDIR, filename);
    fout.open ( outfile );
    if ( fout ) {
        for (i=0; i<n; i++ ) fout << q[i] << char(TAB) << x[i] << char(TAB) << endl;
        fout.flush(); fout.close();
    } else {
        sprintf(msg,"Error opening %s for output...",filename);
        pgm_err(CBS_NOTOPN,msg);
    }
}

/*****************************************************************************
*
*     lma()
*
*   Performs an Levenberg-Marquardt Algorithm fit to the formula defined
*   in ftest() and jftest()
*
*****************************************************************************/
int lma(double* x, double* y, int npoints, double* p, double* info) { if ( DEBUG )
pgm_msg("lma()");
    // allocate working space for the algorithm
    double *work = new double [LM_DIF_WORKSZ(NPARAMS,npoints)];
    double *covar = new double[npoints*npoints];
    double opts[LM_OPTS_SZ];

    opts[0]=LM_INIT_MU; opts[1]=1E-15; opts[2]=1E-15; opts[3]=1E-20;
    opts[4]=LM_DIFF_DELTA;

    if ( USE_JACOBIAN ) {
        dlevmar_der(ftest, jftest, p, y, NPARAMS, npoints, MAXITER, opts, info, work,
covar, static_cast<void*>(x));
    } else {
        dlevmar_dif(ftest, p, y, NPARAMS, npoints, MAXITER, opts, info, work, covar,
static_cast<void*>(x));
    }

    // cleanup
    delete [] work; delete [] covar;
    return 0;
}

/*****************************************************************************
*
*     ftest & jftest
*
*   Tests the next iteration with and without the Jacobian
*
*****************************************************************************/
extern "C" {
    /*
    The fitting function is:
    y(x) = p0 Exp[-(x-p1)^2/(2 p2^2)] + p3 x + p4
    where p0, p1, p2, p3 and p4 are the parameters to be found
    */
    void ftest(double* p, double* y, int nparams, int npoints, void* xx) {
        // first cast xx to double*
        double* x = static_cast<double*>(xx);

        for(int i=0; i<npoints; i++){
            double arg = pow((x[i] - p[1])/p[2],2);
            y[i] = p[0]*exp(-0.5*arg) + p[3]*x[i] + p[4];
        }
    }
```

```
    // The Jacobian of ftest
    void jftest(double*p, double *dyda, int nparams, int npoints, void*xx) {
        // first cast xx to double*
        double* x = static_cast<double*>(xx);

        for(int i=0, j=0; i<npoints; i++){
            double arg = pow((x[i] - p[1])/p[2],2);
            dyda[j++] = exp(-0.5*arg);
            dyda[j++] = p[0]*(x[i]-p[1])/(p[2]*p[2])*exp(-0.5*arg);
            dyda[j++] = p[0]*pow(x[i]-p[1],2)/pow(p[2],3)*exp(-0.5*arg);
            dyda[j++] = x[i];
            dyda[j++] = 1;
        }
    }
}

/*****************************************************************************
*
*     lockSet()
*
*     Sets a write lock so that the same program isn't doing things on two
*     different consoles.
*
*****************************************************************************/
void lockSet(lockstruct& lock) { if ( DEBUG ) pgm_msg("lockSet()");
    char    msg[ERR_MAXLEN];

    if ( lock.set ) { sprintf(msg, "%s already set", lock.name); pgm_msg(msg); }

    lock.status = lock_request(lock.name);

    if (lock.status == LOCK_OK) {
        lock.set = true;
        lock.data.ip_node =
        sprintf(msg, "%s Set", lock.name); pgm_msg(msg);
        return;
    }

    if (lock.status == LOCK_OTHER) {
        lock_read(lock.name, &lock.data);
        sprintf(msg,"%s on CLX%u", lock.name, abs(lock.data.ip_node));
    } else {
        sprintf(msg, "%s failure", lock.name);
    }
    pgm_err(lock.status,msg);
    pgm_msg("VSA Communication inhibited",YELLOW);
    return;
}

/*****************************************************************************
*
*     lockUnset()
*
*     Releases the lock set by the program
*
*****************************************************************************/
void lockUnset(lockstruct& lock) { if ( DEBUG ) pgm_msg("lockUnset()");
    pgm_err( lock_release(lock.name) ,"VSA Communication inhibited");
}

#ifdef LINUX
    static char const * const LcppScrubIssues[] __attribute__((unused)) = {
        "$LcppScrubIssues: fp-flatfiles=addressed $",
        "$LcppScrubIssues: vms-filespec=addressed $",
        "$LcppScrubIssues: vms-cmd-line-string=addressed $",
        };
#endif
```

References

[i] D. Edwards & J. Syphers, An Introduction to the Physics of High Energy Accelerators, § 3.2.3, © 2004, WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

[ii] H. Wiedemann, Particle Accelerator Physics I, Second Edition, § 6.2, © 2003 Springer-Verlag

[iii] M. Gasior & R. Jones, *The Principle and First Results of Betatron Tune Measurement by Direct Diode Detection*, §2 http://doc.cern.ch/archive/electronic/cern/preprints/lhc/lhc-project-report-853.pdf

[iv] A. Jansson, P. Lebrun, R. Pasquinelli, *Experience With the 1.7 GHZ Schottky Pick-ups in the Tevatron*, http://beamdocs.fnal.gov/DocDB/0012/001238/001/THPLT135.pdf

[v] M. Gasior & R. Jones, *The Principle and First Results of Betatron Tune Measurement by Direct Diode Detection*, §3.2 http://doc.cern.ch/archive/electronic/cern/preprints/lhc/lhc-project-report-853.pdf

[vi] General Cable, *RG-58/U Type Coaxial Cable Data Sheet*, http://www.generalcable.com/NR/rdonlyres/4F9763E4-0F09-426F-A58E-7A159BC7549E/0/Pg074_076_RG58U.pdf

[vii] Dennis Barnaal, Analog Electronics for Scientific Application, p. A25, © 1989, Waveland Press, Prospect Heights, IL

[viii] M. Gasior & R. Jones, *The Principle and First Results of Betatron Tune Measurement by Direct Diode Detection*, §3.3 http://doc.cern.ch/archive/electronic/cern/preprints/lhc/lhc-project-report-853.pdf

[ix] CY Tan & D Edstrom, *TeV 3D-BBq Log* (unpublished), 5/24/2007 – 6/8/2007, http://www-bd.fnal.gov/cgi-mach/machlog.pl?nb=ops&action=view&page=33

[x] P. Cameron et al, *Observations of Direct Excitation of the Betatron Spectrum by Mains Harmonics in RHIC*, http://www.agsrhichome.bnl.gov/AP/ap_notes/ap_note_253.pdf

[xi] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. Numerical Recipes in C, p. 683, Cambridge University Press, 2nd edition, © 1992

[xii] Manolis Lourakis, *Levenberg-Marquardt nonlinear least squares algorithms in C/C++*, Institute of Computer Science, Foundation for Research and Technology - Hellas, Heraklion, Crete, Greece, http://www.ics.forth.gr/~lourakis/levmar/

[xiii] A. Jansson, *Effect of coupling on Tevatron 1.7 GHz Schottky tunes*, 2/2/2005, http://beamdocs.fnal.gov/DocDB/0015/001576/002/e17%20coupling.pdf

[xiv] UL_LMA CVS Directory, http://www-ad.fnal.gov/cgi-cvs/cvsweb.cgi/mecca/uls/ul_lma/

[xv] *Agilent Technologies 89400-Series GPIB Command Reference*, Agilent Technologies Part Number 89400-90039, © 2000, Everett, Washington

[xvi] PA4040 CVS Directory, http://www-ad.fnal.gov/cgi-cvs/cvsweb.cgi/mecca/pas/pa4040/

[xvii] D. Edstrom & B. Hanna, TeV E-Log, Tue Oct 28 2008 (unpublished), http://www-bd.fnal.gov/cgi-mach/machlog.pl?nb=tev08&action=view&page=427&anchor=180436&hilite=18:04:36-

Dean Richard Edstrom, Jr.

Box 500, MS306, Batavia IL, 60510
E-mail: edstrom@fnal.gov

Education

Indiana University, Bloomington, IN
Masters of Science, 2004 – 2009 (Anticipated)
Beam Physics and Technology

- Attended classes through the United States Particle Accelerator School program (USPAS) at Fermi National Accelerator Laboratory. Classes in this degree were hosted by universities around the country including The University of California, Berkeley, The University of Wisconsin, Madison, Arizona State University, Boston University, Michigan State University, The University of California at Santa Cruz, and Vanderbilt University.

Gustavus Adolphus College, St. Peter, MN
Bachelor of Arts, 1997 – 2001
Major: Physics Minor: Russian Studies

[redacted]

Employment

Fermi National Accelerator Laboratory
Accelerator Crew Chief, 2009 – Present

- In addition to the responsibilities of an Operator I and Operator II, those of a Crew Chief include:
  - Managing operation of the accelerator complex under the direction of the AD/OPS Dept Head
  - Preparing and delivering briefings and operations performance summaries as required
  - Ensuring that the on-shift log is complete and up to date
  - Functioning as the Area Emergency Supervisor of the Accelerator Division while on duty
  - Delegation of tasks to Operators

Accelerator Operator II, 2002 – 2009

- Promoted to Accelerator Crew Chief on 3/01/2009
- In addition to the responsibilities of an Operator I, those of an Operator II include:
  - Assisting in the training of less experienced Operators (Accelerator Operator I).
  - Performing the duties of Acting Crew Chief as assigned.

Accelerator Operator I, 2001 – 2002

- Promoted to Accelerator Operator II on 12/30/2002
- Responsibilities include
  - Operating and tuning the Accelerator Division's accelerators and beam transport systems as required by the current operating schedule.
  - Executing the first level of troubleshooting and performing minor repair of equipment.
  - Assisting accelerator and beamline physicists in making experimental measurements.
  - Developing and maintaining knowledge and awareness of equipment status and changes.
  - Documenting shift activities and repairs in the logbook thereby helping to maintain an on-shift log of accelerator performance and unusual behavior of equipment.
  - Performing "search and secure" procedure of enclosures to ensure safe operation of the accelerators.
  - Assisting in radiation surveys as required.
  - Performing all duties in accordance with all environmental, health and safety regulations and practices pertinent to this position.

The Gustavus Adolphus College Physics Department (1998 – 2001)

- Programming and physics demonstrations

[redacted]