Sounding-Rocket Studies of Langmuir-Wave Microphysics in the Auroral Ionosphere

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the
degree of
Doctor of Philosophy
in
Physics and Astronomy
by

Micah P. Dombrowski

Dartmouth College
Hanover, New Hampshire

10 March 2016

Examining Committee:

Jim LaBelle (Chair)
Kristina A. Lynch
Robyn M. Millan
Iver H. Cairns
Craig A. Kletzing

Our choicest plans have fallen through,
our airiest castles tumbled over,
because of lines we neatly drew
and later neatly stumbled over.
— Piet Hein, Grooks

Preface

Having made known my plans to leave academia, I've been asked by a few people whether I dislike science, or my field in particular, and whether I regret having gone to graduate school, or determining to go for the PhD. While I think it's fairly common to feel a certain ambivalence towards many things at the end of this process, I do not regret the friends made, the learning, or the experiences I've had, and shared. I have simply decided that there are things I would like to do which would not be possible or as effectively achieved within the framework of academia.

I certainly do not feel this degree was a 'waste of time' in any way. Regardless of whether the flashy piece of paper and extra letters near my name help me in any way, the tools and methods and ways of looking at problems I've learned in the past nine years are universal. Physics has at its heart the paradigm of problem solving...and there are always problems to solve.

It was not till I really started thinking about it that I realized how many thanks I could potentially pass around. There are the obvious people, the big ones, but a thesis is a lot of words and images, and there's a lot of work in and behind it. Everything I create and do and even *think* is the end result of experiences and influences, knowledge and wisdom, built up in sometimes odd or surprising ways, from myriad sources. Some of the people I would like to mention cannot know that I did so, and others would not care if they did know; yet, I feel motivated to make the list fairly exhaustive...

Thank you to my mother, for raising me to be such a nerd, and then putting up with the results. To all of my family all over the world, for their love and support. And to Belle von Woo for keeping me sane and warm, and tolerating my shenanigans.

To Jim LaBelle for his enthusiasm and expertise, and for not giving up on me. Well-met to my colleagues at StingeCo: Chris, Nick, Matt, Xi, Howard, and Spencer—not only would science have been more difficult without you, it also wouldn't have been nearly as entertaining.

My gratitude for help and advice to David McGaw and Mike Trimpi, Bill Hamblen, Richard Brittain, Susan Schwarz, and Terry Kovacs.

Much thanks to Doug Rowland for NASA GSRP mentoring, to Steve Kaeppler for camaraderie and assistance, and to everyone at the NASA Wallops Flight Facility, for making our science things go into space, as well as for being pretty awesome to work with. Thanks also to the support staff at South Pole Station, for endless help with our recalcitrant equipment.

Cheers to all of my Dartmouth friends and cohorts—if suffering shared is suffering halved, then this all can't have been that big a deal...right? Right. Shout-outs to Jess, Jerod, and Julie, Thiago, Nic, Mingyun, and Idan—alas, Ben, we hardly knew ye. Thanks to Damian for dinners and dramatics, and to all my wine and cheese connoisseurs. Thanks to Greg Feiden, for many things, including making this LaTeX template.

Much gratitude to my committee members, for reading this brick, and then making it better. Thanks to all the faculty in the Dartmouth Physics & Astronomy department, for enlightening classes and discussions.

To Franklin & Marshall faculty and friends, especially Ken Krebs and Beth Praton for being amazing undergrad research advisors. To Tim, Brian, Becky, Rory, Elizabeth, and Zach, for making long, homework-filled nights seem short. To the Fry Guy, for being pretty damned awesome.

And of course, thanks to my HACC Lancaster friends, faculty, and coworkers—every journey starts somewhere, and though it was 'only' a quarter of my time in college, my experiences at the Ha'vahd of Pennsylvania still influence everything I do and create—even this thesis.

Special thanks to Stephan Williams, for enlightenment and laughter, for being a dear and lasting friend, and for supporting dastardly schemes.

Long live **The Herodotus Society**!

Love to Gingerbread, for listening when I fall apart—to Arch, for that thing, and Cinna, for words and snacks. To Exto, Jackie, HMC, and Archer, for amusement and consternation, and Ara, for a cheesesteak. To Xaq, for her hard work and nerdery. Hi, Jesin! Thanks to Bolty, for J.

To Beatrix and Butterflies, for focus, and to the Pink One, for having my back.
One needs one's totems, after all...

For being both the oddest and best mixture I've known of thoughtful discussion, absurdity, and amusement, thanks to the Jury, ya crazy sack of lemurs. And of course, my gratitude to He of Many Names, for making it happen, and for fueling the wild-and-nearly-baseless-speculation part of my brain.

To the East Horse, and Scrivener, for endless drama. To Sanguinius, for breadcrumbs, and Dromicosuchus, for a revelation. To Faust, for being a fan, and the unforeseen consequences thereof.

To Quinn, Sigmund, Miko, Isis, Osiris, Faust, Mayet, and Florian, for faithful service.

To Reiska, for never losing touch.

To Linus Torvalds, and Steve Jobs, for operational diversity.

To Notch, for infinity.

To Nintendo, Square, Cyan, and Valve; Tetsuya Mizuguchi, Davey Wreden, and Jonathan Blow—for interactive art.

To Takashi Miike, for The Happiness of the Katakuris.

To Gene Roddenberry, for vision.

To Jamie W. Zawinski and Laura Lemay, for early influences.

To Adam and Jeff, Vir, Thethi, Sev, Erin, and Deth, for personalities.

To Douglas Adams, Terry Pratchett, Steven Brust, Bill Watterson, Jon Rosenberg, and Pete Abrams, for good humor.

To Terry Brooks, Celia S. Friedman, Iain M. Banks, Mercedes Lackey, Connie Willis, Vernor Vinge, Karl Schroeder, Anne McCaffrey, and C. S. Lewis—for worlds.

To Mrs. Beal, for recognition, support, and immersion.

To Bob Ross and Julia Child, for being modern sages.

...and to the Sun and Moon, for life and inspiration.

Contents

List of Tables

List of Figures

Chapter 1

Above the Aurora

Figure 1.1: Aurora seen from the International Space Station (NASA/ESA/ Alexander Gerst https://twitter.com/Astro_Alex/status/505282945272524800).

Earth seems a fantastic place full of magic and wonderment. Alas, as for children growing up, we have, as a species, cast off our notions of magic, and pounded our heads against reality until it gave us deeper knowledge and understanding of the interactions underpinning what once we explained fancifully. So it is with the aurora, both borealis and australis: not for many years have those with any scientific interest been for long able to pass the northern and southern lights off as the work of spirits, ghosts, or gods—excepting, perhaps, if one were to desperately attempt to classify auroral spirits as spirits of gaseous fluorescence.

We have examined the gasses in our atmosphere, and asked why they would ever be so gauche. "Tis not our fault," they declaim, "but that we are so excited by these brutish electron beams."

While it is tempting to simply pass it off as poor taste on the part of these gasses—certainly oxygen is possessed of enough character flaws already, always getting into other elements' business—an expanded examination enlightens everyone, exposing an electric edifice. Elucidating: there exists a grand electric circuit[1], the volumetric bulk of which lives in space, but which exists because Earth and its magnetic field do, and indeed, makes one leg of its journey through Earth's upper atmosphere.

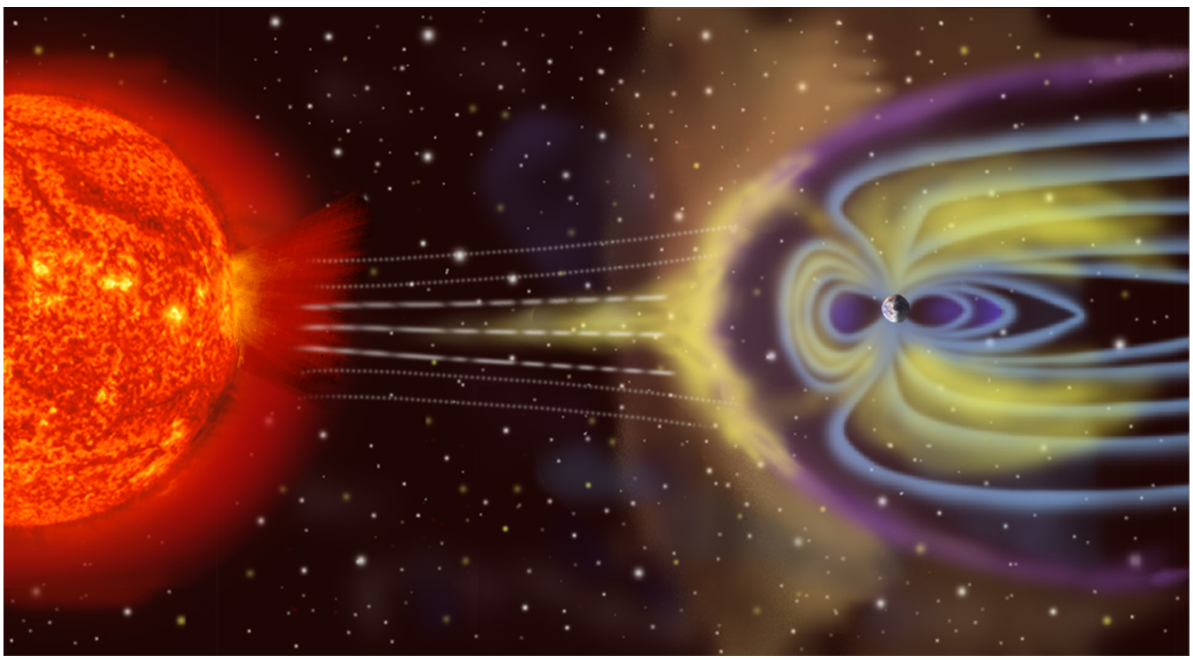

Figure 1.2: An artist's rendition of the solar wind blowing against the Earth's magnetosphere (NASA SDO http://sdo.gsfc.nasa.gov/mission/).

Exists because of, but not solely: as with everything in our general vicinity, our system needs a Sol. As the Sun spins its merry way through the galaxy, it also projects out a panoply of particles both neutral and not, and its very own prodigious magnetic field proceeds along with the plasma part of this play. As far as one can push a comparison of this outflow of solar-wind soup to a river, one then begs leave to similarly portray the Earth and its much more minute magnetic field as a water wheel, capturing a significant fraction of the energy

[1]Not to be confused with its neighboring circuit, which connects the Earth's surface to the ionosphere via lightning.

that impinges on it, and the power output of which drives a myriad of processes throughout Earth-space.

Here then as in Figure 1.2 we have the origin of our electric circuit of interest: the Sun and Earth's magnetic fields grinding both with and against each other (as mercurial moods mandate), providing both energy and direction to a party of particles from both bodies. This uproarious region, defined in the fore by the balance of solar wind pressure versus Earth's magnetic field, and in the aft by the whimsies of chaotic interactions, and being shaped in a general sense along the lines of a teardrop as shown in Figure 1.3, is classified as the Earth's magnetosphere. Of its associated plasmas, the Ionosphere is merely the most earthward march.

Figure 1.4 forthwith portrays but a small fraction of the bevy of chaotic interactions and phenomena which call these regions home; however, we shall in the wake of our projective expansion choose to limit ourselves closer to our original purview: precisely, processes within the auroral ionosphere, though these are of course affected by more distant realms.

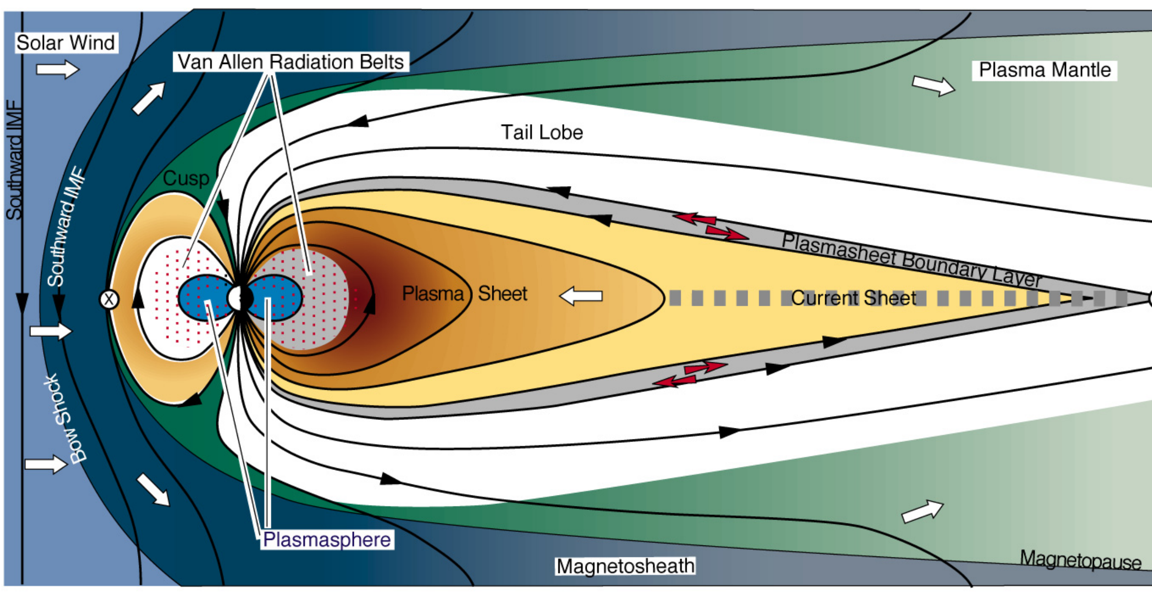

Figure 1.3: A basic diagram of the Earth's magnetosphere, with several features of the field and plasma populations visible (T. W. Hill via http://space.rice.edu/IMAGE/livefrom/sunearth.html).

1.1 PARTICLES!

Far above the the lovely lightshow, we find electrons which move within the confines delineated by Earth's magnetic field, $\boldsymbol{B}$, forming a multipart current system, parts of which are depicted in Figures 1.3 and 1.4. The electrons most important for our current consideration come from a population trapped in the plasma sheet: a long, thin tail of hot, relatively dense

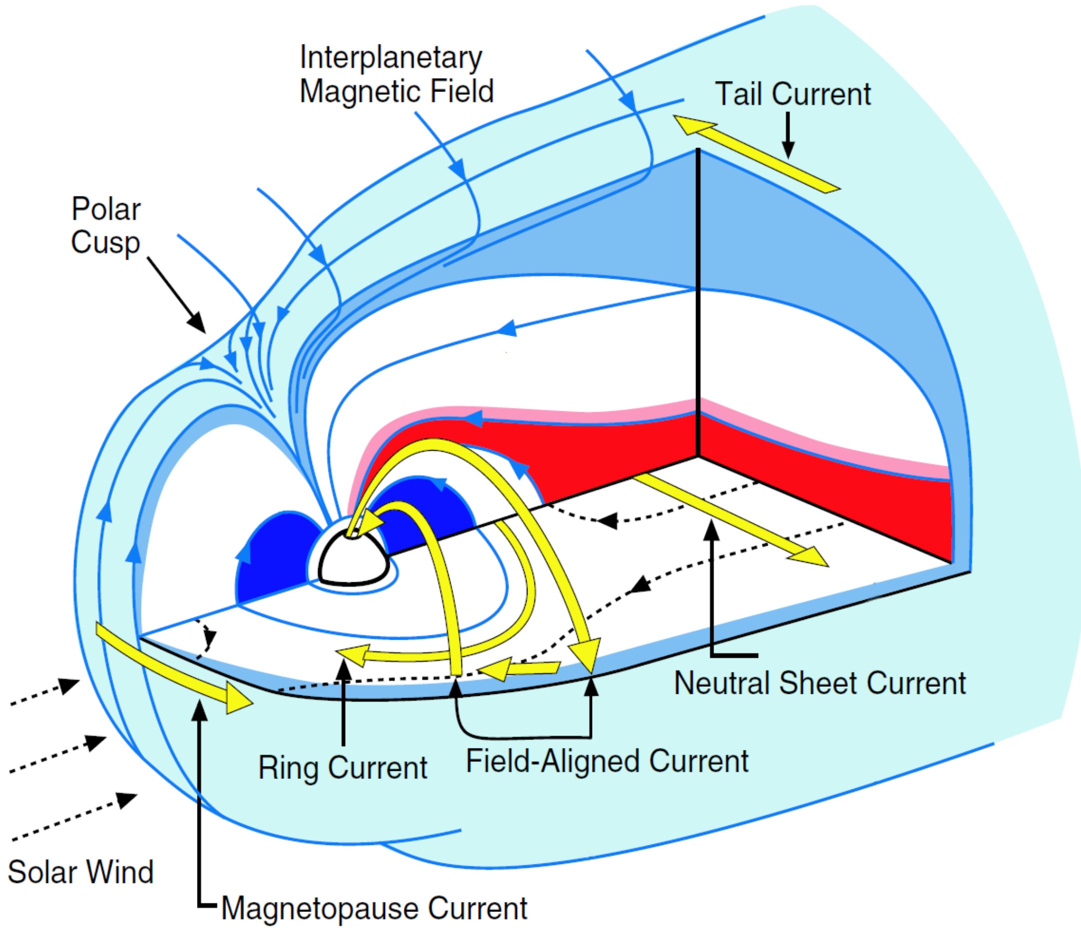

Figure 1.4: A depiction of current systems within the magnetosphere (Adapted from *Keyser et al.* (2005)).

plasma trapped between the two lobes of the tail region of the magnetosphere. Our circuit then travels along the highly conductive magnetic field lines, towards the Earth's magnetic poles—there, as the magnetic field lines continue on into the neutral, insulating lower atmosphere, they first find a nice, lower-resistance region in which to complete the circuit: the higher-density regions of ionosphere. These $\boldsymbol{B}$-parallel or 'field-aligned' currents, also known as Birkeland currents, are the primary source of aurora, though at their end the circuit is completed by the Pederson and Hall currents, which are perpendicular to $\boldsymbol{B}$.

Variations in and interactions between these various currents and fields can create large potential drops along field lines, accelerating electron beams, which pump energy into atmospheric neutrals and plasmas. The subsequent release of some of the energy through photon emission—fluorescence—is what we call aurora. This is, however, only a small part of the story. While the result most obvious to the eye is auroral illumination, improvement of instrumentation and techniques has refined our understanding of these processes from a vague knowledge of the bulk energy transfer, to current knowledge of fluctuations in the electron-beam source regions on smaller and smaller temporal and spatial scales, as well as the presence of small-scale density variations, gradients, and cavities.

From Figure 1.4 it is clear that there are both upward and downward current regions, but it is the lower-latitude, upward currents which are most important for generating the aurora, for it is here that electrons are accelerated downward in order to fulfill those current needs. The activity varies for several reasons, including the direction of the solar wind with respect to the Earth's field and the overall intensity of solar activity. With high levels of solar activity comes increased occurrence of magnetic reconnection, in which field lines in the sun-facing, or 'dayside' magnetosphere open to solar field lines at the bow shock, and then wrap around to the tail/nightside, releasing significant amounts of energy and often injecting both magnetosheath (external solar) and magnetospheric plasma into the current system, increasing auroral activity.

A significant place where reconnection and magnetosheath plasma have effects is in the magnetic cusps (see Figure 1.3), where the closed field lines which lead the bowshock trade off with the open field lines that form the outer fringe of the tail. Here, the field leaves a gap which even at quiet times can offer some amounts of solar plasma direct access to the ionosphere.

The energy being pumped into the ionospheric plasma has many more effects observable by more diverse instrumentation; in particular, radio-wave and charged-particle detectors. While the electron beams travel, already oscillating around their field lines at the cyclotron frequency $f_c = \frac{|q|B}{m_e}$, they may interact with the ionospheric plasma as its density increases to the F-region peak, then falls. The ionosphere is a perfect environment for the generation of various plasma and electromagnetic waves.

1.2 WAVES!

Any plasma with incoming energy is a natural environment for the generation and propagation of a menagerie of waves. The great variety of waves we can detect in-situ via satellites and sounding rockets can, in theory, serve as probes of regions in the plasma; however, a thorough understanding of wave generation, mode conversion, and transit is required. Small-scale frequency-time wave signals are generated by plasma structures and modified in-transit by other structures, and so specific types of wave phenomena can be used to probe their associated small-scale plasma structures.

While both the electrons and ions have a part in plasma dynamics, when studying the electron component of a plasma, it is often convenient to treat the ions as an immobile background (effectively infinite mass). This is a fine approximation to make as long as the frequencies being studied are high enough that the ion motion is negligable.

In an electron plasma, the simplest and most fundamental waves are oft referred to as 'Langmuir waves', after Irving Langmuir, who first detected them in laboratory discharge plasmas (*Langmuir* 1928). This is an oscillation in the electrons which is most easily visualized in a 'cold' plasma, where the thermal velocities are ignored. Then, given a stationary population of electrons with a background of fixed ions, any displacement of the electrons will be restored by the Coulomb force, setting up an oscillation at the 'plasma frequency' $\omega_{pe} = \sqrt{\frac{n_e e^2}{m_e \varepsilon_0}}$, where n_e is simply the density of electrons in the plasma. In a cold plasma, the Langmuir wave is a standing wave.

In a warm plasma, when we take the electron thermal motion into account, the electron pressure is an additional restoring force, and the frequency is determined by $\omega^2 = \omega_{pe}^2 + 3k^2 v_{e,th}^2 = \omega_{pe}^2 + \frac{3k_B T_e}{m_e} k^2$ with $v_{e,th} = \sqrt{k_B T_e / m_e}$ the electron thermal velocity and $\bar{k}$ the wave vector. This propagating wave can be visualized as a compressional wave along the direction of propagation, or along the magnetic field line in a magnetized plasma.

As Langmuir waves directly interact with both the local plasma and impinging electron beams, they are highly sensitive to the motion and instability of particle populations in the upper atmosphere. Thus, greater understanding of Langmuir wave generation, damping, propagation and interaction is essential to understanding ionospheric plasma and its temporal and spatial variations. A greater understanding of the proportionality and behavior of Langmuir modes through direct measurement can also yield insights into plasma heating and energetic particle sources in such environments.

As the primary instability that occurs in beam-plasma interactions, Langmuir waves are ubiquitous in space physics; thus, any localized study of Langmuir waves may yield insights into processes and interactions in other environments. There are too many examples to cover them all, but we review below four case studies which serve to illustrate the range of phenomena they are involved in, the diversity of locations, and how important they are in space physics.

Langmuir waves exist in diverse regions, sometimes because of subtle interactions. While prior observations of Langmuir waves in Earth's foreshock correctly predicted their presence as arising from the two-stream instability, *Filbert and Kellogg* (1979) examined data from the Imp 6 satellite, and found that the required double-peaked distribution is generated by time-of-flight effects from interactions at the bow shock. Figure 1.5 depicts this process, in which the tip of the bow shock acts as a reflector and source of particles, modifying the distribution enough to yield a second peak in a small region. It was also shown that the resultant growth rates were high enough that some additional mechanism is required to stabilize the distribution in so short a time that the generative unstable distribution was not observable with the Imp 6 instrumentation. They found that a mixture of wave-wave interaction and quasi-linear relaxation in different spatial regions was sufficient.

Lin et al. (1981) examined a particularly strong type III solar radio burst observed 17 February 1979 on the ISEE 3 satellite. They find Langmuir-wave growth consistent with a bump-on-tail distribution generated by dispersion, as in Figure 1.6. They also found that additional mechanisms were required to fit the observations: that wave growth must be limited by nonlinear processes such as wave-wave interaction or the emission and collapse of soliton structures, and also that the distribution did not plateau as expected, implying that waves are being removed from resonance with the positive slope in the distribution. They also note the impulsive character to these Langmuir waves, typical of type III bursts.

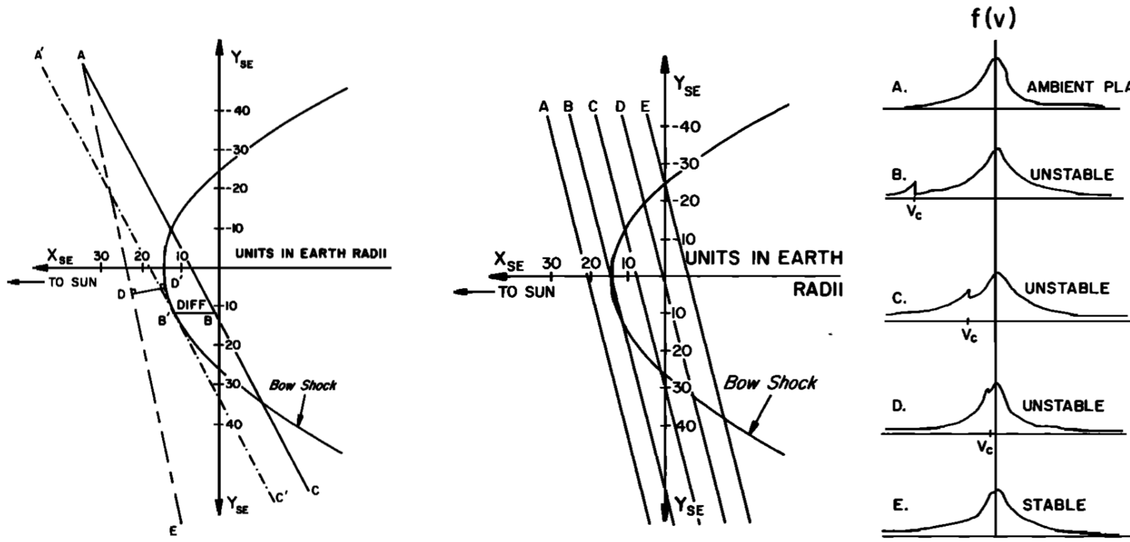

Figure 1.5: The process described by *Filbert and Kellogg* (1979), by which a two-stream instability is produced in the Earth's foreshock. Left, in the narrow region between tangential field line A' and its parallel A, the bow shock acts as a reflector of electrons traveling down solar field lines. Right shows the effect on the distribution, creating a double-peaked distribution in a small region.

As Langmuir waves are a key avenue of energy-release for their generative electron beams, the processes which mediate beam-plasma energy transfer are key to how and why they are able to propagate long distances. Because of this, a thorough understanding of Langmuir wave growth, interaction, and decay processes can aid in understanding electron beam transit and the evolution and relaxation of beam distributions.

Gurnett et al. (1981a) examined data from the 1979 Voyager 1 flyby of Jupiter, finding several examples of intense Langmuir waves upstream of Jupiter's bow shock, generated by the mechanism described by *Filbert and Kellogg* (1979). In addition to allowing an independent measurement of the electron density profile (*Gurnett et al.* 1981b), the high sampling rate of the Voyager wave instrument with respect to the local ω_{pe} allowed for observations of fine structure in the wave data, including the variable amplitude modulation shown in Figure 1.7. This is interpreted as the result of beating between the Langmuir waves and sideband waves created due to parametric interactions, possibly involving background ion-acoustic waves. Further, they found some evidence of soliton-like structures being created and subsequently collapsing, as suggested by *Lin et al.* (1981).

More recent observations, such as by the STEREO spacecraft, find no evidence of soliton collapse in the solar wind (*Graham et al.* 2012). Instead, stochastic growth appears to mediate energy flow to and from relaxed electron beams, with fine and inhomogeneous density structures playing a crucial role in the foreshock (*Malaspina et al.* 2009). The need for three-dimensional instrumentation is also becoming clear, as significant wave power connected to type III solar radio bursts has been observed both parallel and perpendicular to the background magnetic field (*Malaspina and Ergun* 2008; *Graham and Cairns* 2014).

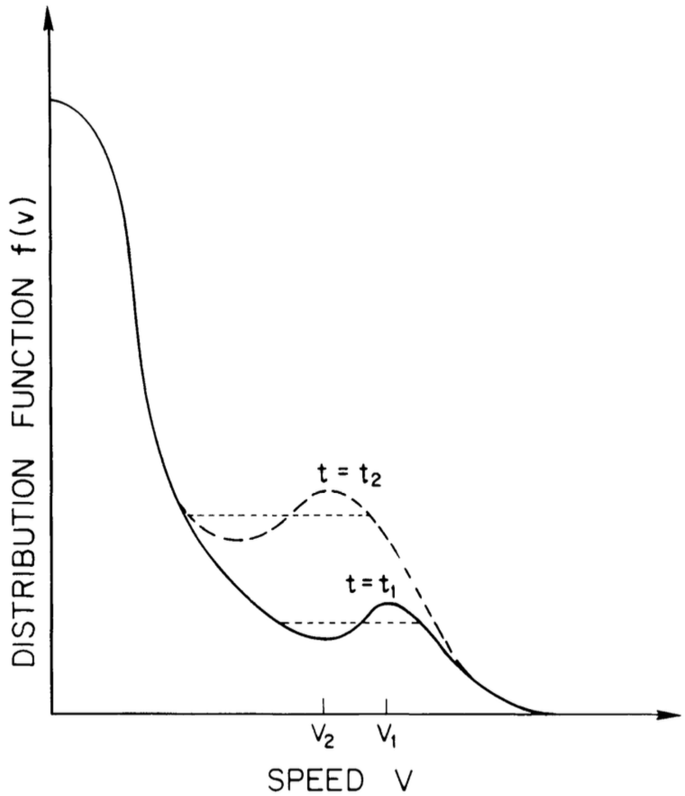

Figure 1.6: A model of a bump-on-tail distribution generated by a dispersive electron beam, presented by (*Lin et al.* 1981). The highest-energy beam electrons arrive first (time t_1), creating the bump and its wave-generating positive slope. The waves should eventually plateau the region, though as slower electrons arrive (t_2) the peak may shift to lower velocities first.

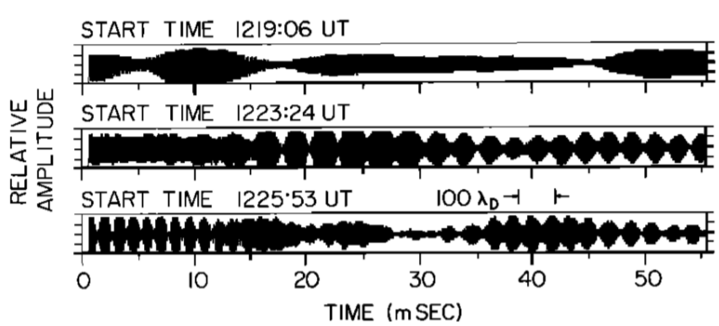

Figure 1.7: The Langmuir wave amplitude modulation observed by Voyager 1 (*Gurnett et al.* 1981a).

As with particle instrumentation, radio wave receivers and data returns have constantly improved. Langmuir waves, as well as other HF waves, have been observed to be, not continuous, but sporadic and at times highly structured. *McAdams* (1999) studied a number of high-frequency phenomena which occur near intense Langmuir-wave activity. 'Bands' are monochromatic, narrow-band emissions seen in underdense ($\omega_{pe} < \omega_{ce}$) regions, and speculated to originate as Langmuir waves generated in regions of highly variable plasma density, and subsequently converted in higher-density regions to long-lived, freely propagating whistler-mode waves. Chirps are impulsive emissions with a time-varying, primarily decreasing frequency, and are consistent with the 'spike' emissions observed via satellite and reported by *Beghin et al.* (1981), as well as with a proposed mechanism wherein Langmuir waves generated in density cavities are converted to Z-mode waves. More interesting still are effects arising from excitation of Langmuir waves in regions characterized by pre-existing electron density fluctuations. *McAdams et al.* (2000) show that the 'bands' observed by *McAdams and LaBelle* (1999) result when the scale size of the density fluctuations is comparable to the Langmuir wavelength. In this case, the Langmuir waves form discrete eigenmodes, and the frequency spacing of these eigenmodes predicted by theory matches that of the Langmuir wave 'bands'. Similar phenomena are reported by *Malaspina et al.* (2012) for Langmuir-wave related solar wind emissions. Doppler shifting due to the high solar-wind speed makes the eigenmodes harder to discern, but their presence is convincingly demonstrated through fitting the theory to the data.

While Langmuir waves have been observed and characterized in many environments, this work seeks to address two particular gaps which still exist. One of these lies in the realm of three-dimensional, high sample-rate measurements, which are nearly unknown, though *Malaspina and Ergun* (2008) have observed some three dimensional structure via the STEREO spacecraft. Such measurements have the potential to speak to questions regarding wave-wave interactions and Langmuir-wave spatial structure. The other gap lies in the area of wave-particle correlation, which aims to directly examine the microphysics of the wave-particle interactions which both generate Langmuir waves and damp them away. Only a few such instruments have been reported on in the literature, and with a very limited number events observed in each case (*Gough et al.* 1990; *Boehm et al.* 1994; *Ergun et al.* 1998; *Kletzing et al.* 2005).

Chapter 2 covers related development efforts to improve a new Receiver/Digital Signal Processor system which has a frequency range appropriate to study ionospheric Langmuir waves, as well as many other space plasma phenomena. An iteration of this system was flown on the CHARM-II sounding rocket mission, and is currently deployed at several sites for ground-based auroral radio studies. Included is a detailed description of an in-situ high time- and frequency- resolution observation of auroral roar.

In Chapter 3 we describe results from the ongoing collaborative Dartmouth College/University of Iowa wave-particle correlation study, an iteration of which was flown on the CHARM-II mission. These results are analyzed and discussed, and then a numerical simulation is implemented to evaluate the plausibility of two related theories explaining what was observed.

Chapter 4 details a three-dimensional study of the bursty nature of Langmuir waves observed on the TRICE mission. A theory explaining these is reviewed and tested, involving wave-wave interaction with oblique electrostatic waves.

Finally, Chapter 5 summarizes our results, and suggests future continuations of this work, while the Appendices shall record the major programmatic codes created for this work, and attempt to document their usage and quirks.

Bibliography

Beghin, C., J. L. Rauch, and J. M. Bosqued, Electrostatic plasma waves and hf auroral hiss generated at low altitude, *J. Geophys. Res.*, *94*(A2), 1359–1378, 1981.

Boehm, M., G. Paschmann, J. Clemmons, H. Höfner, R. Frenzel, M. Ertl, G. Haerendel, P. Hill, H. Lauche, L. Eliasson, and R. Lundin, The tesp electron spectrometer and correlator (f7) on freja, *Space Sci. Rev.*, *70*, 509–540, 1994.

Ergun, R. E., J. P. McFadden, and C. W. Carlson, Wave-particle correlator instrument design, in *Measurement Techniques in Space Plasmas: Particles*, *Geophysical Monograph*, vol. Particles, pp. 325–331, American Geophysical Union, American Geophysical Union, 1998.

Filbert, P. C., and P. J. Kellogg, Electrostatic noise at the plasma frequency, *J. Geophys. Res.*, *84*(A4), 1369–1381, 1979.

Gough, M. P., P. J. Christiansen, and K. Wilhelm, Auroral beam-plasma interactions: Particle correlator investigations, *J. Geophys. Res.*, *95*(A8), 12,287–12,294, 1990.

Graham, D. B., and I. H. Cairns, Dynamical evidence for nonlinear langmuir wave processes in type iii solar radio bursts, *J. Geophys. Res.*, *119*(4), 2430–2457, doi:10.1002/2013JA019425, 2014.

Graham, D. B., I. H. Cairns, D. R. Prabhakar, R. E. Ergun, D. M. Malaspina, S. D. Bale, K. Goetz, and P. J. Kellogg, Do langmuir wave packets in the solar wind collapse?, *J. Geophys. Res.*, *117*(A09), A09,107, doi:10.1029/2012JA018033, 2012.

Gurnett, D. A., J. E. Maggs, D. L. Gallagher, W. S. Kurth, and F. L. Scarf, Parametric interaction and spatial collapse of beam-driven langmuir waves in the solar wind, *J. Geophys. Res.*, *86*(A10), 8833–8841, 1981a.

Gurnett, D. A., F. L. Scarf, W. S. Kurth, R. R. Shaw, and R. L. Poynter, Determination of jupiter's electron density profile from plasma wave observations, *J. Geophys. Res.*, *86*(A10), 8199–8212, 1981b.

Keyser, J. D., M. W. Dunlop, C. J. Owen, B. U. Ö. Sonnerup, S. E. Haaland, A. Vaivads, G. Paschmann, R. Lundin, and L. Rezeau, Magnetopause and boundary layer, *Space Sci. Rev.*, *118*(1), 231–320, doi:10.1007/s11214-005-3834-1, 2005.

Kletzing, C. A., S. R. Bounds, J. LaBelle, and M. Samara, Observation of the reactive component of langmuir wave phase-bunched electrons, *Geophys. Res. Lett.*, *32*(L05106), doi:10.1029/2004GL021175, 2005.

Langmuir, I., Oscillations in ionized gasses, *Proceedings of the National Academy of Sciences*, *14*(8), 627–637, 1928.

Lin, R. P., D. W. Potter, D. A. Gurnett, and F. L. Scarf, Energetic electrons and plasma waves associated with a solar type iii radio burst, *Astrophys. J.*, *251*(1), 364, 1981.

Malaspina, D. M., and R. E. Ergun, Observations of three-dimensional langmuir wave structure, *J. Geophys. Res.*, *113*(A12108), 2008.

Malaspina, D. M., B. Li, I. H. Cairns, P. A. Robinson, Z. Kuncic, and R. E. Ergun, Terrestrial foreshock langmuir waves: Stereo observations, theoretical modeling, and quasi-linear simulations, *J. Geophys. Res.*, *114*(A12), A12,101, doi:10.1029/2009JA014493, 2009.

Malaspina, D. M., I. H. Cairns, and R. E. Ergun, Antenna radiation near the local plasma frequency by langmuir wave eigenmodes, *Astrophys. J.*, *755*(45), 2012.

McAdams, K., and J. LaBelle, Narrowband structure in hf waves above the electron plasma frequency in the auroral ionosphere, *Geophys. Res. Lett.*, *26*(13), 1825–1828, 1999.

McAdams, K. L., Sounding rocket based investigations of hf waves in the auroral ionosphere, Ph.D. thesis, Dartmouth College, 6127 Wilder Laboratory, Hanover, NH 03755, 1999.

McAdams, K. L., R. E. Ergun, and J. LaBelle, Hf chirps: Eigenmode trapping in density depletions, *Geophys. Res. Lett.*, *27*(3), 321–324, 2000.

Chapter 2

An Autonomous Receiver/Digital Signal Processor for Wave Experiments: The Dartmouth Rx-DSP

2.1 Introduction

Instabilities in space plasmas produce waves in a wide range of frequencies and bandwidths, with a large variety of time signatures, detectable both in situ and remotely. Detector technologies include inductive loops for magnetic fields, double probes for electric fields, and Langmuir probes for plasma density. For receivers, the ideal wave analysis instrument would involve a direct high-frequency analog-to-digital (ADC) sampling of the output of a given detector or antenna, with the highest possible sampling rate and bit depth. While technology has advanced in recent years to allow continuous sampling at 20 MHz or beyond, it is often not feasible to use such techniques directly, due to limited data transmission and storage capabilities.

Furthermore, it is often desirable to record wave data from multiple detectors simultaneously, e.g. from spatially separated or orthogonal antennae. Such measurements can allow detection of wave polarization and propagation directions. Simultaneous sampling requires a high degree of ADC sample synchronization across multiple receivers, and results in even greater demands on data storage and transmission systems, rendering direct simultaneous sampling even less attractive.

Data storage and transmission limitations are at their most severe on spacecraft, and therefore many innovative solutions have come out of that community. For example, the Cluster satellites, launched in 2000, included the Wide-Band plasma investigation (WBD). This instrument was capable of downconverting in selected frequency bands, removing the need for storage of samples at twice the Nyquist rate (*Gurnett et al.* 1997). Another example is the

Waves instrument onboard the Van Allen Probes (formerly RBSP), launched August 2012, which is similar to the WBD, but also allows for dynamic Fast Fourier Transforms (FFTs) and data compression (*Kletzing et al.* 2013). The receiving system most similar to the subject of this chapter is the Radio Receiver Instrument (RRI) on board the e-POP payload of the Canadian CASSIOPE satellite. The RRI directly samples four probes at 40 MHz and then performs on-board signal processing (*James et al.* 2015).

The Dartmouth Receiver/Digital Signal Processor (Rx-DSP) represents another recent development effort to address these issues. As a digital downsampling receiver, it can transmit wave data within a specific band or set of bands within the 0 to 33 MHz range. The data can be sampled either continuously or in bursts, allowing for fine-grained customization of the transmission data rate. In addition, the Rx-DSP boards are designed for cross-receiver sample synchronization to within 2 nanoseconds. The Rx-DSP is set apart by its autonomous capabilities with remote reprogrammability, high maximum sample rate, and myriad options for data transmission. The generalized nature of the instrument front-end allows for use with a wide range of detector hardware. It also allows for a variety of both spacecraft-borne and ground-based applications, as discussed below.

Section 2.2, describes the current iteration of the Dartmouth Rx-DSP hardware, and Section 2.3 explains the naming convention for individual deployments. Section 2.4 provides an overview of the firmware used on the onboard programmable DSP. Finally, Section 2.5 presents three examples of applications of this system to space physics, with case studies of one rocket mission and two ground-based detectors.

2.2 Hardware

The Rx-DSP is a low-cost analog-to-digital receiver and signal processor board, designed for use in both ground and space scenarios, and specifically engineered for cross-board sample-synchronized acquisition. The use of purpose-specific receiver components allows for a significant shortening of system development cycles as compared to an FPGA-based solution, by removing programming, testing, and debugging complexities; however, the specific components chosen for the Rx-DSP platform maintain appreciable flexibility in the field. The detailed architecture of the boards has sounding rocket flight history from instruments produced at the University of Iowa. The current generation of boards have been tested for reliable operation at temperatures from 0 to 50 C—more extreme ground environments require external regulation, such as placement in insulated or heated boxes, whereas sounding rockets are warmed on the launch pad, and flights are not long enough for cooling to be a concern. While the Rx-DSP design could be extended for high-radiation space environments, this has not been a goal of current development efforts. Data acquisition systems incorporating the Rx-DSP are easily crafted for autonomous operation with no external command and control, transmitting results via a number of protocols. Figure 2.1 shows a picture of the topmost Rx-DSP board in a stack of two—a configuration used in several applications. The data flows through the board as in Figure 2.2, going through asynchronous Receive, Processing, and Transmit stages.

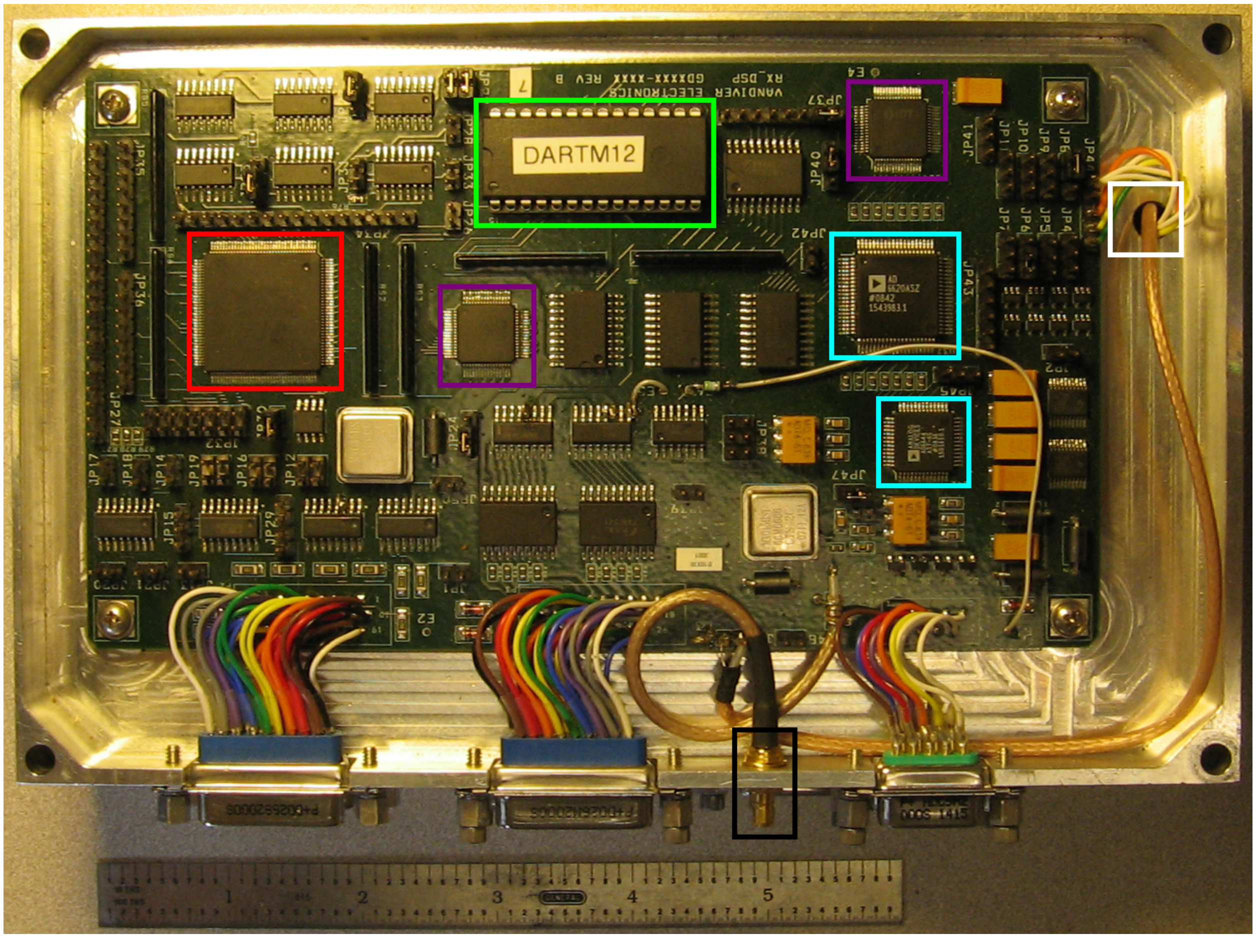

Figure 2.1: A photograph of the top board of an Rx-DSP stack ready for a rocket flight, with 6 inch ruler for scale. Highlighted are the SMB signal input (black), cross-board synchronization lines (white), AD6644 & AD6620 signal processors (cyan), IDT72285 FIFOs (purple) TMS320C542 programmable processor (red), and the program-code EEPROM (green).

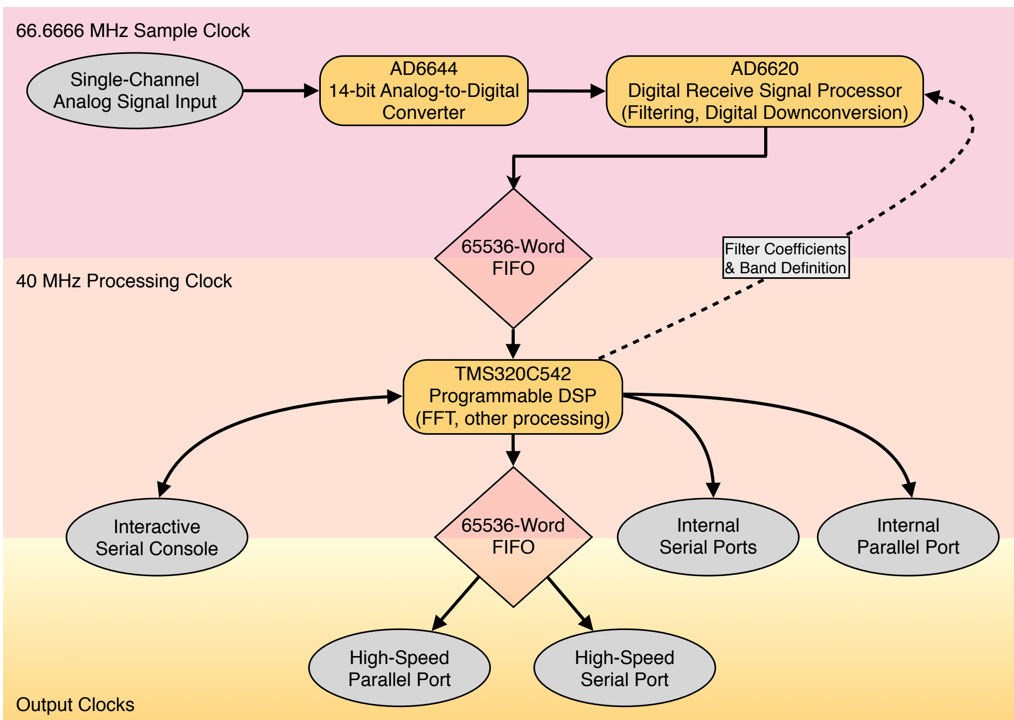

Figure 2.2: A diagram depicting the major parts of the Rx-DSP hardware, and the data flow between them, with the dashed line indicating command/control and solid lines indicating data or both. The colored background boxes indicate which systems are controlled by which clocks.

The Receive stage takes a balanced analog signal with a maximum 1 volt peak-to-peak amplitude, fed to the input of an Analog Devices AD6644 ADC, which samples at 66.6666 MHz with 14-bit resolution, yielding a 33.3333 MHz Nyquist frequency, 74 dB signal-to-noise ratio (SNR), and 100 dB spurious-free dynamic range. There is no built-in filtering, and an input bandwidth of 250 MHz, allowing for undersampling downconversion; thus, each application requires customized front-end pre-amplifiers and filters for band-limiting and antialiasing. The outputs of the 6644 are linked directly to an Analog Devices AD6620 programmable digital Receive Signal Processor (RSP). This processor performs quadrature frequency translation, and then decimates and filters the incoming signal through three stages, yielding a band with width, center, and filter characteristics defined by a table of values and filter coefficients. The RSP can further improve the SNR, and the total system performance and frequency response will be unique to each application, determined by the preamplifiers, filters, and cabling used. The quadrature data is output from the RSP as 16-bit words, with In-phase and Quadrature (I and Q) words being interleaved, and each word is accompanied by a bit which indicates whether a given sample is an I or Q word. This relatively low-frequency, 17-bit data is then stored to an 18-bit Integrated Device Technology IDT72285 First-In First-Out (FIFO) buffer.

The receive FIFO output is accessible to a Texas Instruments (TI) TMS320C542 Digital Signal Processor. This processor has a number of useful built-in peripherals, runs on an external clock (generally set for 40 MHz operation), has 10 kilowords of built-in RAM, and can access up to 16 KB of program code and tables from an external PROM. In many deployments, this DSP acts only as a data router and packager, adding headers and/or synchronization information before passing the data onwards. However, by loading custom software to this processor, a variety of real-time, streaming data processing effects are achievable, such as FFTs and various types of compression, though no such deployments will be shown in the case-studies herein.

After all desired data processing steps are complete, the data in memory can follow a number of output paths. First, the data can be sent at high speed to a second IDT72285 FIFO. The outputs of this FIFO are accessible to high-speed serial and parallel LVDS outputs, at any speed up to the full quadrature data rate. A second option can exploit one of two serial ports available on the TMS320C542: a buffered serial port that allows efficient data transfer at standard RS-232 speeds, and a time-division multiplexed port that allows multiple boards to share one serial link. A third option makes use of a parallel Host Port Interface that allows the DSP to connect to an external device at high speeds (up to 8 MBps). Finally, a fourth possibility is to wire and program the Rx-DSP to allow dropping to a single-line interactive serial console, through which a user can trigger single acquisitions, read data, configure settings, or remotely re-burn the firmware EEPROM.

In many use cases, the DSP is able to spend idle time in a low-power mode, significantly reducing the average power draw of the Rx-DSP board—without detailed optimization, the power draw per Rx-DSP is approximately 1.5 W. The flexibility in configuration, coding, and data output allow for a wide range of receiver setups. In addition, the AD6620 is designed to allow for sample synchronization across chips, and the Rx-DSP boards are designed to allow the sample clocks and RSPs to be synchronized as well, using short (< 10 cm) jumper wires which pass the clock and AD6620 synchronization lines between boards. This allows for the development of multi-board setups for wave-polarization measurements and direction finding.

2.3 Nomenclature

Each individual deployment of Rx-DSP hardware requires custom hardware for input refinement, power, command input, and data output. For ease of referral, each Rx-DSP system may be referred to as an **A**utonomous **R**x-DSP **C**luster (ARC), with a prefix signifying current data collection intent. The current set of prefixes are arrayed below:

1. P - Polarization
2. F - Fine Structure
3. M - Multi-Band
4. I - Imaging/Direction-Finding
5. S - Spectrum Analyzing

The other element which is generally different in each ARC is the firmware loaded by the TMS320C542 processor.

2.4 Firmware Overview

The limited RAM on the TMS320C542 processor is shared between loaded programs and data, requiring careful management of program size and data storage. The programs used are all hand-coded in TI DSP Assembly, except for the FFT module, which is based on code from the TI DSP C Library. The default mode upon power-up has the DSP load its program code from the onboard PROM and then commence execution.

The program code developed at Dartmouth for rocket and ground-based application is modular, but all implementations follow a general structure outlined in Figure 2.3. After initializing the C542 and AD6620 hardware, the AD6620 acquisition is started, and data is loaded into RAM by the C542. For continuous high-speed data acquisition, the AD6620 may be left 'on'; however, when only discrete data blocks are required, power usage can be cut significantly by halting acquisition between blocks.

Once the data is in RAM, any number of processing steps can apply, limited only by available RAM and processing time. In the simplest case the data is untouched. In the most complex case currently coded, 1024-word FFTs are performed on incoming data. For most cases, the data is next encapsulated in a synchronization framework, which includes sync words, sampling-specification headers, and frame counters. The processed data is next prepared for output.

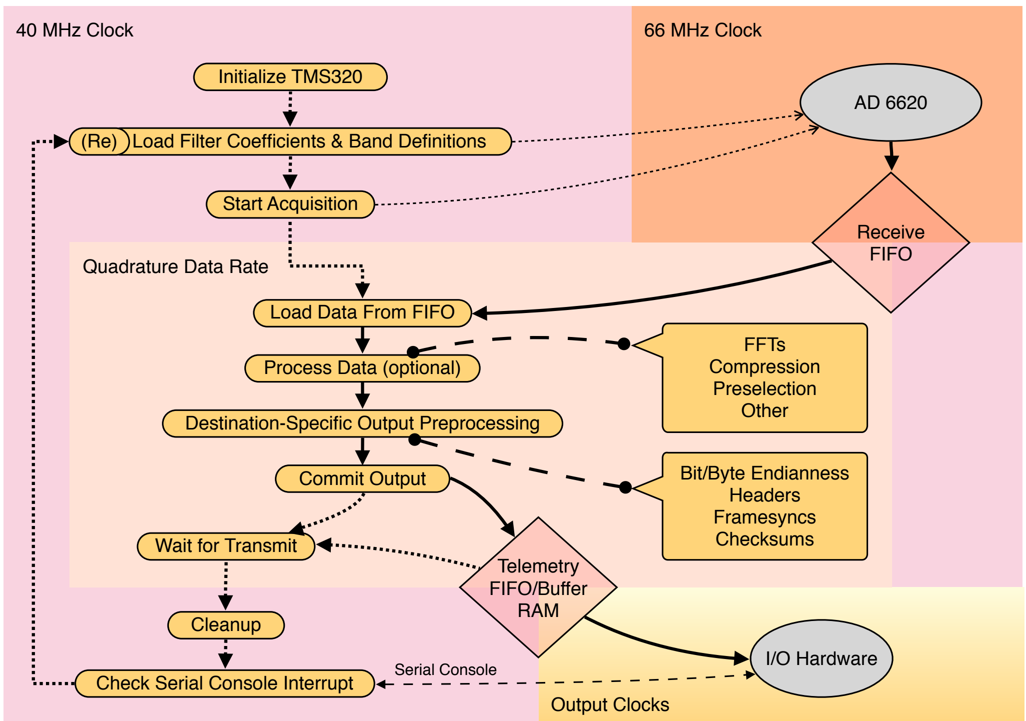

Figure 2.3: A diagram depicting a generalized program flow for the Rx-DSP assembly code. Dashed lines indicate command/control flow, while solid lines include data as well. Color backgrounds show which parts of the code run at the given clock rates, with FIFOs and wait cycles allowing for asynchronous operation. The two callout boxes show modularized routines in the codebase, some, all, or none of which may be used by a given ARC.

Data handling for output varies widely, depending on final destination, DSP setup, and output hardware. To output to the high-speed serial or parallel systems, data is merely copied into the output FIFO and then read out via rocket telemetry or PC USB hardware. For output involving the C542 chip's built-in peripherals, various preprocessing steps may be required, including downsampling, data subset selection, endianness conversion, and the addition of extra sync data and headers. The most efficient C542 peripheral for data output is the Buffered Serial Port, which merely requires that its rotating buffer is periodically filled. All other peripherals require that each byte be individually preloaded. In either case, data loading can either be handled by fixed software loops, or can be interrupt driven.

A special case for input and output on the DSP is the software serial console interface. This link allows a PC with a standard RS-232 serial port to connect to the C542, which can be switched into the serial console mode via an external toggle. The console allows for single acquisitions, direct editing of program code in RAM, modifications to the AD6620 setup, and for the uploading and burning of new PROM files for permanent changes.

2.5 Case Studies

2.5.1 CHARM-II – Rocket-Borne Application

Auroral roar is a natural ionospheric radio emission characterized by a relatively narrow-banded structure centered at frequencies near multiples of the electron cyclotron frequency. It is most frequently observed by ground-based radio receivers, but has also been seen by satellites (*James et al.* 1974; *Benson and Wong* 1987; *Bale* 1999). The intense electrostatic upper-hybrid waves which are the source of auroral roar have been detected by a sounding rocket, but hitherto not the auroral roar itself (*Samara et al.* 2004). Detailed ground-based studies have shown that many instances of roar are not singular emissions, but rather contain intricate fine structures visible on high-resolution frequency-time plots (*LaBelle et al.* 1995; *Shepherd et al.* 1998b). Further studies have determined that the lowest harmonic of roar seen on the ground ($2f_ce$) is left-hand elliptically polarized with respect to the local magnetic field (*Shepherd et al.* 1997), while there have been observations of higher harmonics being either left or right-hand polarized (*Sato et al.* 2012). It is theorized that roar originates as upper-hybrid plasma waves above the ionospheric 'F peak', converting through linear or nonlinear processes into propagating electromagnetic waves (*Shepherd et al.* 1998a; *Yoon et al.* 2000; *Ye et al.* 2007), and the HIBAR and Porcupine sounding rockets may have encountered regions of such plasma waves (*Carlson et al.* 1987).

The Correlation of High-Frequency and Auroral Roar Measurements (CHARM-II) auroral sounding rocket carried the second successful deployment of the Rx-DSP hardware. On the CHARM-I mission the Rx-DSPs returned approximately 1-2 minutes of data from exposed, partially deployed electric-field probes, before these probes sheared off due to catastrophic payload failure. The CHARM-II mission was launched from the Poker Flat Research Range near Fairbanks, AK, at 9:49 UT/22:46 MLT on 16 February 2010, reaching an apogee of 802 km. The payload carried a two-board FP-ARC, each receiver digitizing the differential

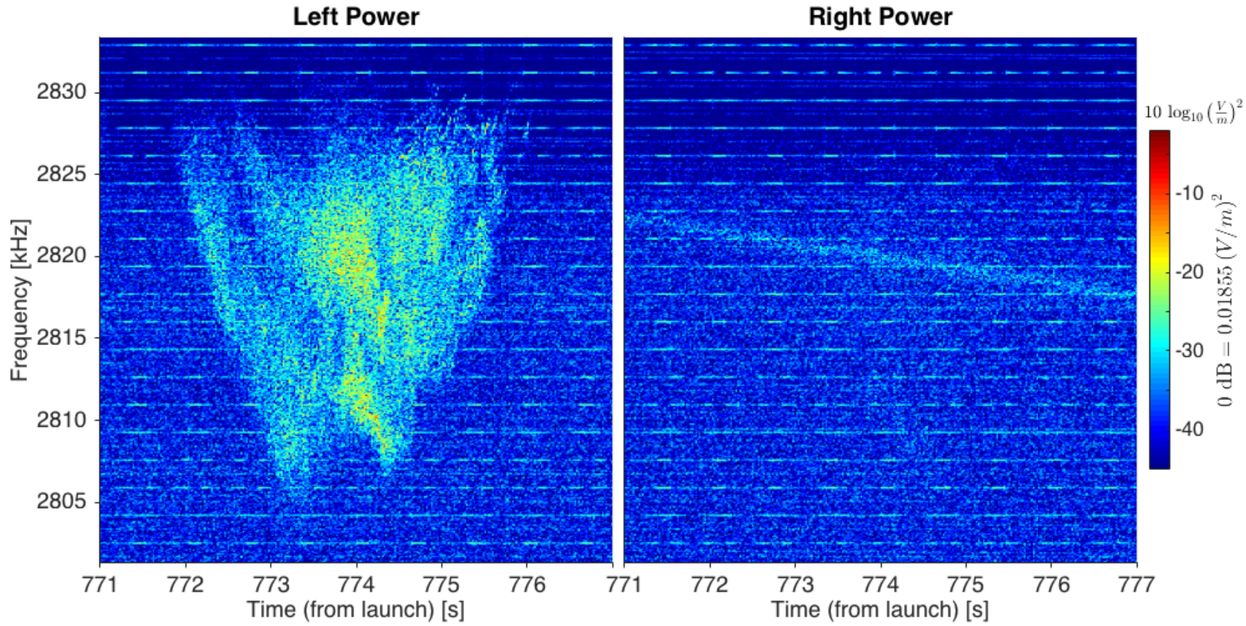

Figure 2.4: Power spectra of Rx-DSP data from CHARM II, recombined to yield left- and right-circularly polarized powers. The line of power with decreasing frequency seen in the righthand plot is an interference line of unknown origin which exists through much of the flight, and has been seen on other flights.

voltage between two 2.5 cm spherical aluminum probes, with the two probe sets positioned perpendicular to each other in the plane orthogonal to the rocket's spin axis, which was oriented parallel to the geomagnetic field. The Rx-DSPs were in a simple downsampling mode, adding short headers and outputting through the high-speed telemetry FIFO and LVDS serial link. The data rate was set to maximally utilize two S-band telemetry links, transmitting downsampled data in a 333 kHz band centered at 2.67 MHz. As the payload nominally had its spin axis aligned with the Earth's magnetic field, $\boldsymbol{B}$, the Rx-DSPs in this configuration effectively yielded a picture of the projection of electric-field wave activity onto the plane perpendicular to $\boldsymbol{B}$ within the designated band.

The CHARM-II FP-ARC yielded the first in-situ observation of auroral roar with both high time resolution and polarization data. Figure 2.4 shows spectrograms over a 298 to 330 kHz band from 771 to 777 seconds after launch, corresponding to 548 to 536 km altitude on the downleg of the flight. The color scale represents the power of righthand circularly polarized signals (a) and lefthand circularly polarized signals (b), with polarizations being with respect to $\boldsymbol{B}$.

Figure 2.4 was generated using a technique described by *LaBelle and Treumann* (1992), adapted from *Kodera et al.* (1977). Given time series data corresponding to two perpendicular, transverse components of a field, as from the measured X and Y components from the Rx-DSPs, a spectral power can be estimated for lefthand and righthand circular wave polarization by recombining the complex Fast Fourier Transforms (FFT) of the time series, according to

$$\begin{aligned} FFT_L &= FFT_X + \imath \times FFT_Y, \\ \text{and} \quad FFT_R &= FFT_X - \imath \times FFT_Y. \end{aligned}$$

For the CHARM II data, the two perpendicular quadrature signals are detected in situ, and transmitted to ground via payload telemetry systems. In post-flight processing, the data is FFTed, and then recombined to yield the estimated left and righthand powers shown in Figure 2.4.

Figure 2.4 clearly establishes that the observed waves are lefthand polarized. Not only does this confirm the ground-level observations of *Shepherd et al.* (1997), it expands upon it, as the high time and frequency resolution makes it clear that the individual fine structures are all lefthand polarized. *Sato et al.* (2015) have performed a similar analysis for ground-level $4f_{ce}$ roar emissions. The lefthand polarization of these waves is consistent with various generation theories, especially those put forth by *Yoon et al.* (2000).

2.5.2 South Pole Station – Ground-Based Application

South Pole Station (SPS) lies on the Antarctic Plateau thousands of kilometers from commercial and other broadcast activities associated with population centers. As a result, the station is very favorable for studies of radio emissions of natural origin, and hosts a variety of radio receivers at ELF to HF frequencies, complemented by other geophysical diagnostics such as all-sky cameras, photometers, and flux-gate magnetometers. Significant observations at VLF (*Martin* 1960; *Chevalier et al.* 2007), LF-MF (*LaBelle et al.* 2005; *Ye et al.* 2006; *Yan et al.* 2013; *Broughton et al.* 2014), and HF (*Rodger and Rosenberg* 1999; *Patterson et al.* 2001) have been made at the station.

Hence, it was a natural decision to deploy the Rx-DSP to the South Pole. In January 2012 Dartmouth installed a PF-ARC at SPS, consisting of two Rx-DSP boards wired to perform synchronized sampling. Two 40 m^2 loop antennas perpendicular to each other, supported by a 10 m mast, were constructed about 1 km from the station. Figure 2.5a shows these antennas. The planes of the loops are perpendicular to the ground and to each other, providing highest sensitivity to waves coming from overhead, and allowing right- and left-hand polarization to easily be distinguished from the phase relation between the signals. The ARC, a duplicate of that shown in Figure 2.5b was programmed for continuous sampling of a 330-kHz band centered on 515 kHz. Data were offloaded to a PC through the Rx-DSP parallel LVDS link via a pair of QuickUSB high-speed USB data acquisition modules, and stored on an array of 2 TB hard drives. Spectral and cross-spectral analysis of the signals on the Linux computer determined power and polarization of all signals exceeding the noise level. All computer hardware as well as the ARC were housed in an insulated box as in Figure 2.5c, designed to retain waste heat, keeping them within their operating temperature range after installation in the unheated V8 science vault at SPS.

Figure 2.6 shows spectrograms recorded by this ARC on two days in 2013: July 8 and August 2. In both cases, five minutes of data from one of the two signals are shown, and the data come from within 1.5 hours of magnetic midnight, which occurs at 03:35 UT at

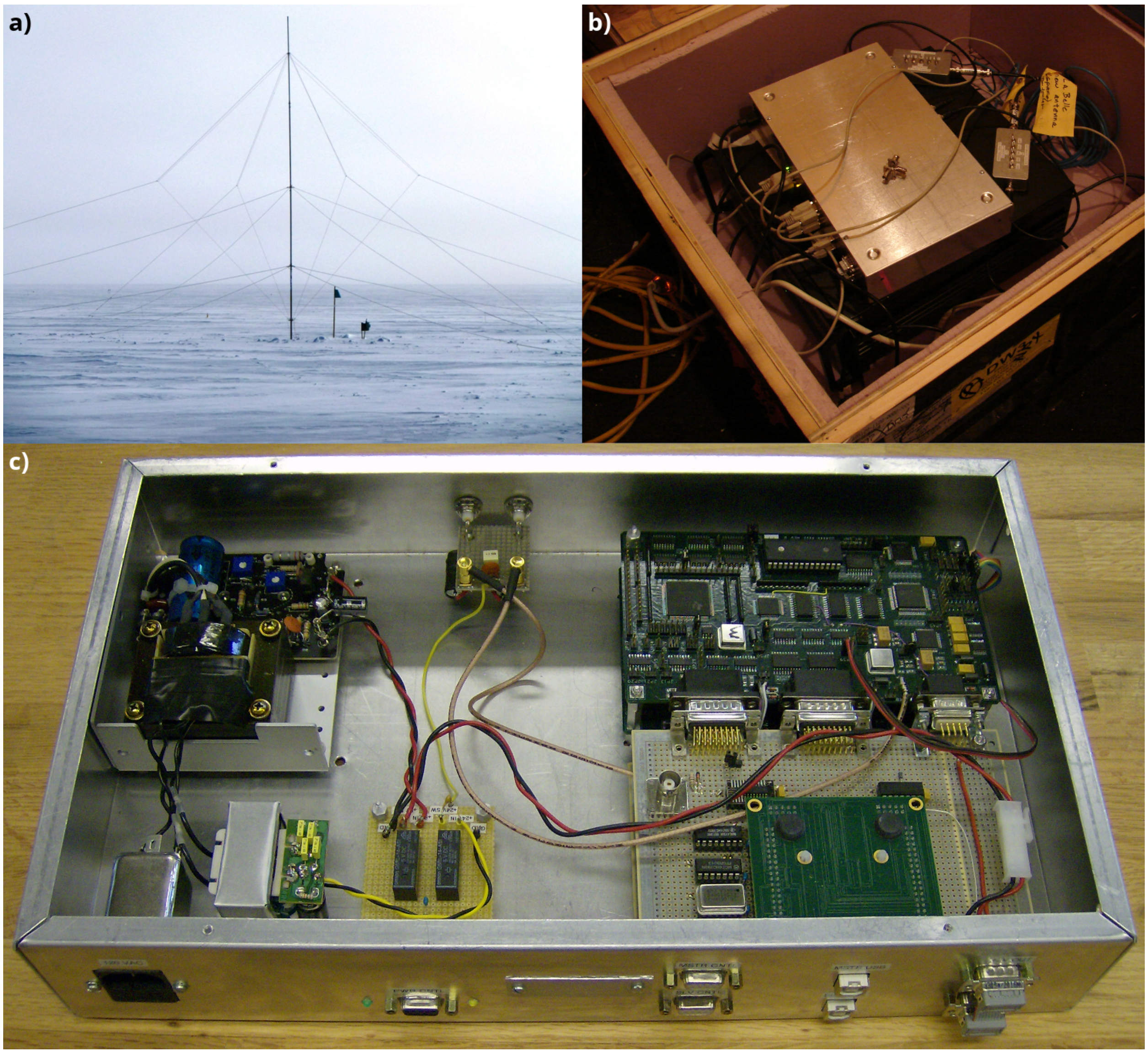

Figure 2.5: Photos of the various components of the South Pole Station PF-ARC. Top left shows the crossed-loop antenna with a 30 ft mast, and the pre-amplifier at the base. Top right shows the receiver box, data-acquisition PC, and various other equipment within an insulated box (covered when in operation). Bottom shows a lab-bench photo of a PF-ARC, with two vertically stacked, sample-synchronized Rx-DSP boards and adjoined QuickUSB breakout boards on the right side.

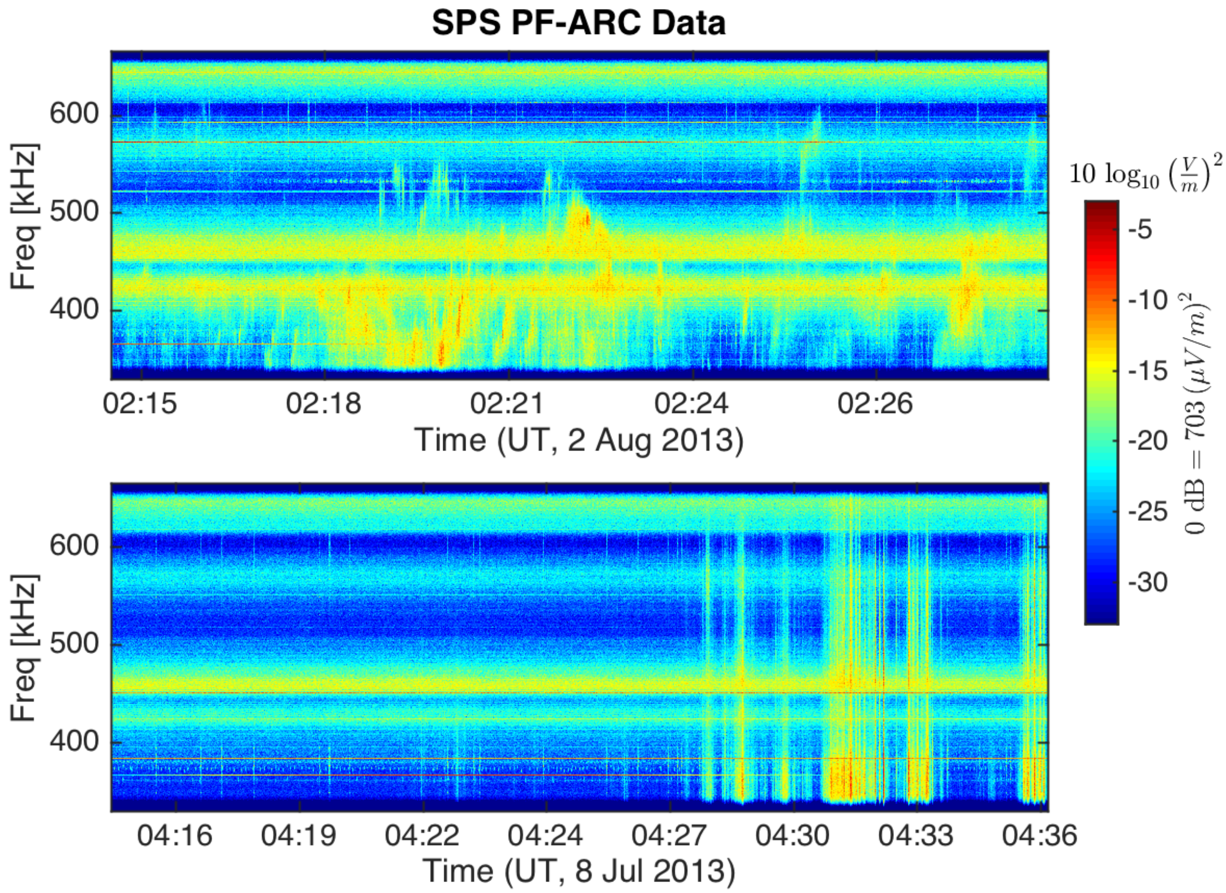

Figure 2.6: Results from the PF-ARC at South Pole Station. The upper spectrogram shows fine structures in signals which appear similar to Auroral Kilometric Radiation, while the lower plot shows an example of auroral hiss, for comparison.

South Pole. In both spectrograms, sharp decreases in the signal power spectral density near the band edges show the effectiveness of the digital filtering in the RX-DSP which defines the bandwidth. Despite the remoteness of South Pole Station, activities at the station lead to strong interference lines, most prominently at 450-460 kHz and 640-650 kHz in each spectrogram and somewhat more weakly at 570-580 kHz and 420-430 kHz.

However, between these interference lines, both spectrograms show evidence of natural radio emissions of auroral origin. The bottom panel, from July 8, 2013, shows a phenomenon called auroral hiss (*Makita* 1979; *Sazhin et al.* 1993; *LaBelle and Treumann* 2002). The high resolution Rx-DSP data show that at LF the hiss consists of impulsive emissions appearing as vertical lines on the spectrogram.

The top panel, from August 2, 2013, shows a phenomenon called 'AKR-like emissions' (*LaBelle and Anderson* 2011; *LaBelle et al.* 2015). This phenomenon is characterized by complicated fine frequency structure consisting of rising and falling tones with typical slopes of hundreds of Hz per second. These features qualitatively resemble those observed in outgoing

X-mode auroral kilometric radiation (AKR) detected with satellite-borne receivers at great distances from Earth (*Gurnett and Anderson* 1981). As pointed out by *LaBelle et al.* (2015), the strong resemblance between this phenomenon and AKR, combined with the stark differences between it and the auroral hiss shown in the top panel of Figure 2.6, forms a powerful argument for a connection between the ground-level AKR-like emissions and the outgoing AKR observed in space.

Due to the success of these observations, further experiments are planned with the Rx-DSP at South Pole. For example, in Summer-Fall 2014 and Summer 2015, the South Pole ARC was operated during anticipated conjunctions between it and Cluster satellites, with the Cluster wave instrument tuned to the same frequency band, in hopes of detecting identical fine structure in ground and in space. Furthermore, as described above, an S-ARC which can perform live spectrum analysis is being installed in Automatic Geophysical Observatories. These autonomous digital receivers in the low-noise Antarctic environment show promise to make important advances in understanding radio waves of auroral origin.

2.5.3 Sondrestrom Research Facility – Ground-Based Application

The Sondrestrom Research Facility lies on the southwest coast of Greenland near Kangerlussuaq, at 66.99° N 309.06° E and is home to numerous instruments for researching Earth's upper atmosphere. These include an incoherent scatter radar, allsky imagers, riometers, magnetometers, and various radio receivers. The MI-ARC at this site consists of a trio of sample-synchronized Rx-DSPs. Input to these comes from five loop antennae: one reference, two situated 50 m from this along lines perpendicular to each other, and two more at 400 m from reference along the same lines. The antennas are arrayed in a small valley approximately 1 km from the station. The three-board MI-ARC is installed in an unheated vault next to the reference antenna, with the receiver itself in a heated, insulated box. The only connection from the vault to the station is a single coaxial cable, which carries both the serial digital output of the ARC, and DC voltage that powers the ARC. The entire array is calibrated at installation and after any major system changes or repairs, through observation of analog reference signals with known strengths and physical source positions.

The ARC triggers relays to switch between the 50 m and 400 m antenna pairs when digitizing signals above and below 1 MHz, respectively. The DSPs are set for discrete sampling of 750 kHz bands, with the receivers rotating through a set of four center frequencies (475, 1225, 1975, and 2725 kHz) approximately once per second. The data are offloaded through the buffered serial port, interleaved via a hardware serial multiplexer, and then transmitted via RS-232 serial link to a remote PC.

To compute the direction of arrival for incoming signals, the three resultant data streams are combined pairwise through cross-spectrum analysis, and averaged over eight 128 or 512-bin FFT ensembles. Then, given calibration data and knowledge of the antenna layout and cable lengths, the phase delays of the resulting spectra can be used to calculate the direction of arrival of high-coherence signals.

Figure 2.7 shows an example of such an analysis for 14 Sep 2013, using signals from 1-1.5

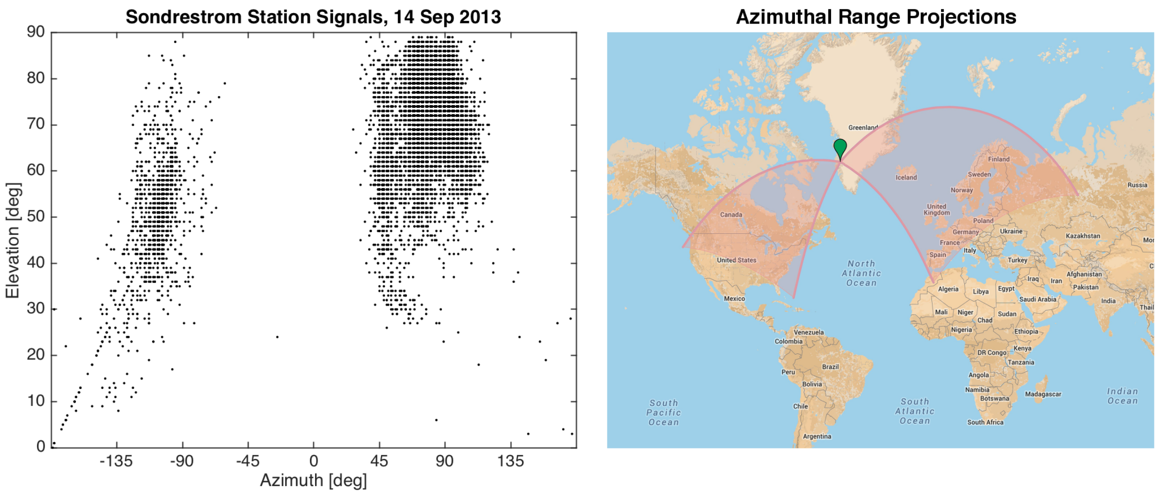

Figure 2.7: Proof of functionality for the MI-ARC at Sondrestrom Station. To the left, an elevation vs. azimuth scatter plot (elevation from the horizon, azimuth in degrees from true north) of high-coherence points for 14 Sep 2013, showing two clear clusters of points. To the right, we project the azimuthal ranges of the two clusters onto a map, implying that the clusters correspond to signals transmitted from Europe and North America [Map data ©2015 Google, INEGI].

MHz with coherence greater than 0.95. The scatter plot above shows elevation vs. azimuth for over 10,000 signals, where elevation is degrees off the horizon and azimuth is degrees from true north. Note that various instrumental uncertainties yield about a 5% uncertainty for each point. The accompanying map shows the approximate azimuthal extent of the two clusters of points. It is clear that the signals detected originate from the directions of North America and Europe. One curiosity is the extension of North American signals to lower elevations, which implies sensitivity to more distant signals. This may be due to atmospheric inhomogeneities or field-of-view anisotropy.

These results establish that the Sondrestrom receiver array/MI-ARC produces accurate direction finding with high time and frequency resolution. The system is resource-efficient, operating autonomously and remotely via a single 1 km coaxial data/power cable. Further data collection for science purposes is ongoing.

2.6 Summary

The Rx-DSP is a flexible platform for high-frequency geophysical data acquisition. ARCs are able to be crafted for autonomous operation in extremely remote regions, for low power draw, and for a wide variety of data transmission rates and media. In particular, the potential for on-board data analysis, reduction, selection, and compression allows for optimal use of low-bandwidth telemetry systems. Additional deployments are already underway, and future revisions of this platform should allow for even more diverse uses.

Thanks to J. C. Vandiver for hardware and firmware development and maintenance, and to Spencer Hatch for presenting this instrument at the 2015 Conference on Measurement Techniques in Solar and Space Physics.

Bibliography

Bale, S. D., Observation of topside ionospheric mf/hf radio emission from space, *Geophys. Res. Lett.*, *26*(6), 667–670, 1999.

Benson, R. F., and H. K. Wong, Low-altitude isis 1 observations of auroral radio emissions and their significance to the cyclotron maser instability, *J. Geophys. Res.*, *92*(A2), 1218–1230, 1987.

Broughton, M. C., J. LaBelle, and P. H. Yoon, A new natural radio emission observed at south pole station, *J. Geophys. Res.*, *119*(1), 566–574, doi:10.1002/2013JA019467, 2014.

Carlson, C. W., R. E. Ergun, A. J. Mallinckrodt, and G. Haerendel, Observations of intense electron bernstein wave emissions in an auroral plasma (unpublished manuscript), 1987.

Chevalier, M. W., W. B. Peter, U. S. Inan, T. F. Bell, and M. Spasojevic, Remote sensing of ionospheric disturbances associated with energetic particle precipitation using the south pole vlf beacon, *J. Geophys. Res.*, *112*(A11), 11,306, doi:10.1029/2007JA012425, 2007.

Gurnett, D. A., and R. R. Anderson, *The Kilometric Radio Emission Spectrum: Relationship to Auroral Acceleration Processes*, *Geophysical Monograph*, vol. Physics of Auroral Arc Formation, AGU, 1981.

Gurnett, D. A., R. L. Huff, and D. L. Kirchner, The wide-band plasma wave investigation, *Space Sci. Rev.*, *79*, 195–208, 1997.

James, H. G., E. L. Hagg, and L. P. Strange, Narrowband radio noise in the topside ionosphere, in *Non-Linear Effects in Electromagnetic Wave Propagation*, no. 138 in Conference Pre-Prints, North Atlantic Treaty Organization, Advisory Group for Aerospace Research & Development, 7 Rue Ancelle, 92200, Neuilly Sur Seine, France, 1974.

James, H. G., E. P. King, A. White, R. H. Hum, W. H. H. L. Lunscher, and C. L. Siefring, The e-pop radio receiver instrument on cassiope, *Space Sci. Rev.*, 2015.

Kletzing, C. A., W. S. Kurth, M. Acuna, R. J. MacDowall, R. B. Torbert, T. Averkamp, D. Bodet, S. R. Bounds, M. Chutter, J. Connerney, D. Crawford, J. S. Dolan, R. Dvorsky, G. B. Hospodarsky, J. Howard, V. Jordanova, R. A. Johnson, D. L. Kirchner, B. Mokrzycki, G. Needell, J. Odom, D. Mark, J. R. F. Pfaff, J. R. Phillips, C. W. Piker, S. L. Remington, D. Rowland, O. Santolik, R. Schnurr, D. Sheppard, C. W. Smith, R. M. Thorne, and J. Tyler, The electric and magnetic field instrument suite and integrated science (emfisis) on rbsp, *Space Sci. Rev.*, *179*, 127–181, doi:10.1007/s11214-013-9993-6, 2013.

Kodera, K., R. Gendrin, and C. de Villedary, Complex representation of a polarized signal and its application to the analysis of ulf waves, *J. Geophys. Res.*, *82*(7), 1245–1255, 1977.

LaBelle, J., and R. R. Anderson, Ground-level detection of auroral kilometric radiation, *Geophys. Res. Lett.*, *38*(4), doi:10.1029/2010GL046411, 2011.

LaBelle, J., and R. A. Treumann, Poynting vector measurements of electromagnetic ion cyclotron waves in the plasmasphere, *J. Geophys. Res.*, *97 (A9)*(13), 789–797, 1992.

LaBelle, J., and R. A. Treumann, Auroral radio emissions, 1. hisses, roars, and bursts, *SSR*, *101*(3, 4), 295–440, 2002.

LaBelle, J., M. L. Trimpi, R. Brittain, and A. T. Weatherwax, Fine structure of auroral roar emissions, *J. Geophys. Res.*, *100*(A11), 21,953–21,959, 1995.

LaBelle, J., A. T. Weatherwax, M. Tantiwiwat, E. Jackson, and J. Linder, Statistical studies of auroral mf burst emissions observed at south pole station and at multiple sites in northern canada, *J. Geophys. Res.*, *110*(A2), doi:10.1029/2004JA010608, 2005.

LaBelle, J., X. Yan, M. C. Broughton, S. Pasternak, M. P. Dombrowski, R. R. Anderson, H. U. Frey, A. T. Weatherwax, and Y. Ebihara, Further evidence for a connection between auroral kilometric radiation and ground-level signals measured in antarctica, *J. Geophys. Res.*, *120*(3), 2061–2075, doi:10.1002/2014JA020977, 2015.

Makita, K., *VLF-LF Hiss Emissions Associated with Aurora*, *Memoirs of National Institute of Polar Research: Aeronomy*, vol. 16, National Institute of Polar Research, 1979.

Martin, L. H., Observations of 'whistlers' and 'chorus' at the south pole, *Nature*, *187*, 1018–1019, doi:10.1038/1871018a0, 1960.

Patterson, J. D., T. P. Armstrong, C. M. Laird, D. L. Detrick, and A. T. Weatherwax, Correlation of solar energetic protons and polar cap absorption, *J. Geophys. Res.*, *106*(A1), 149–163, doi:10.1029/2000JA002006, 2001.

Rodger, A. S., and T. J. Rosenberg, Riometer and hf radar signatures of polar patches, *Radio Science*, *34*(2), 501–508, doi:10.1029/1998RS900005, 1999.

Samara, M., J. LaBelle, C. A. Kletzing, and S. R. Bounds, Rocket observations of structured upper hybrid waves at fuh = 2fce, *Geophys. Res. Lett.*, *31*(22), 2004.

Sato, Y., T. Ono, N. Sato, and Y. Ogawa, First observations of $4f_{ce}$ auroral roar emissions, *Geophys. Res. Lett.*, *39*(7), 2012.

Sato, Y., A. Kadokura, Y. Ogawa, A. Kumamoto, and Y. Katoh, Polarization observations of $4f_{ce}$ auroral roar emissions, *Geophys. Res. Lett.*, *42*(2), 249–255, 2015.

Sazhin, S. S., K. Bullough, and M. Hayakawa, Auroral hiss: a review, *Planetary and Space Science*, *41*(2), 153–166, 1993.

Shepherd, S. G., J. LaBelle, and M. L. Trimpi, The polarization of auroral radio emissions, *Geophys. Res. Lett.*, *24*(24), 3161–3164, 1997.

Shepherd, S. G., J. LaBelle, R. A. Doe, M. McCready, and A. T. Weatherwax, Ionospheric structure and the generation of auroral roar, *J. Geophys. Res.*, *103*(A12), 29,253–29,266, 1998a.

Shepherd, S. G., J. LaBelle, and M. L. Trimpi, Further investigation of auroral roar fine structure, *J. Geophys. Res.*, *103*(A2), 2219–2229, 1998b.

Yan, X., J. LaBelle, G. Haerendel, M. Spasojevic, N. L. Bunch, D. I. Golden, H. U. Frey, and A. T. Weatherwax, Dayside auroral hiss observed at south pole station, *J. Geophys. Res.*, *118*(3), 1220–1230, doi:10.1002/jgra.50141, 2013.

Ye, S., J. LaBelle, and A. T. Weatherwax, Further study of flickering auroral roar emission: 1. south pole observations, *J. Geophys. Res.*, *111*(A7), doi:10.1029/2005JA011271, 2006.

Ye, S., J. LaBelle, P. H. Yoon, and A. T. Weatherwax, Experimental tests of the eigenmode theory of auroral roar fine structure and its application to remote sensing, *J. Geophys. Res.*, *112*(A12304), doi:10.1029/2007JA012525, 2007.

Yoon, P. H., A. T. Weatherwax, and J. LaBelle, Discrete electrostatic eigenmodes associated with ionospheric density structure: Generation of auroral roar fine frequency structure, *J. Geophys. Res.*, *105*(A12), 27,589–27,596, 2000.

Chapter 3

Wave-Particle Correlation in the Auroral Ionosphere: CHARM-II

3.1 Introduction

Langmuir waves, also known as electron plasma waves, are one of the most fundamental properties of a plasma, having been first observed in discharge plasmas in the early days of plasma physics (*Langmuir* 1928). They result from the interaction of electron beams with plasmas and hence are ubiquitous in space plasmas, including, for example, the solar wind, where they generate radio bursts (*Lin et al.* 1981), and planetary foreshocks (*Gurnett et al.* 1981; *Filbert and Kellogg* 1979) and auroral ionospheres (*Kintner et al.* 1995; *Boehm* 1987; *McAdams* 1999; *Samara* 2005), where they mediate energy transfer between the beam and thermal plasmas. Langmuir waves can generate nonlinear structures of fundamental interest to plasma physics, as well as linear eigenmode effects in inhomogeneous plasmas (*McAdams et al.* 2000; *Ergun et al.* 2008). Due to their significance and abundance in the space environment, Langmuir waves are a subject of current study; particularly with regard to their eigenmode structures (*Malaspina et al.* 2012), three-dimensional effects (*Malaspina and Ergun* 2008; *Dombrowski et al.* 2012), and wave-particle correlations (*Ergun et al.* 1991b,a; *Muschietti et al.* 1994; *Kletzing et al.* 2005).

Particle correlation experiments have proven to be an effective way to probe wave-particle interaction physics in space plasmas. A detailed theory of expected results from such instruments regarding Langmuir waves is given by *Kletzing and Muschietti* (2006). The phase bunching of the electrons in the field of the wave can be considered as a superposition of two components, a ‘resistive’ component which is in phase with the wave electric field and represents energy transfer either from wave to particles or vice versa, and a ‘reactive’ component which is in quadrature phase with the wave field and is a signature of electrons trapped in the wave. An early version flown on a sounding rocket in auroral plasma determined a strong correlation between beam electrons and Langmuir/upper hybrid wave electric fields over a several hundred second interval (*Gough et al.* 1990). A wave particle correlation experiment was flown on the Freja spacecraft (*Boehm et al.* 1994). *Ergun et al.* (1998) flew

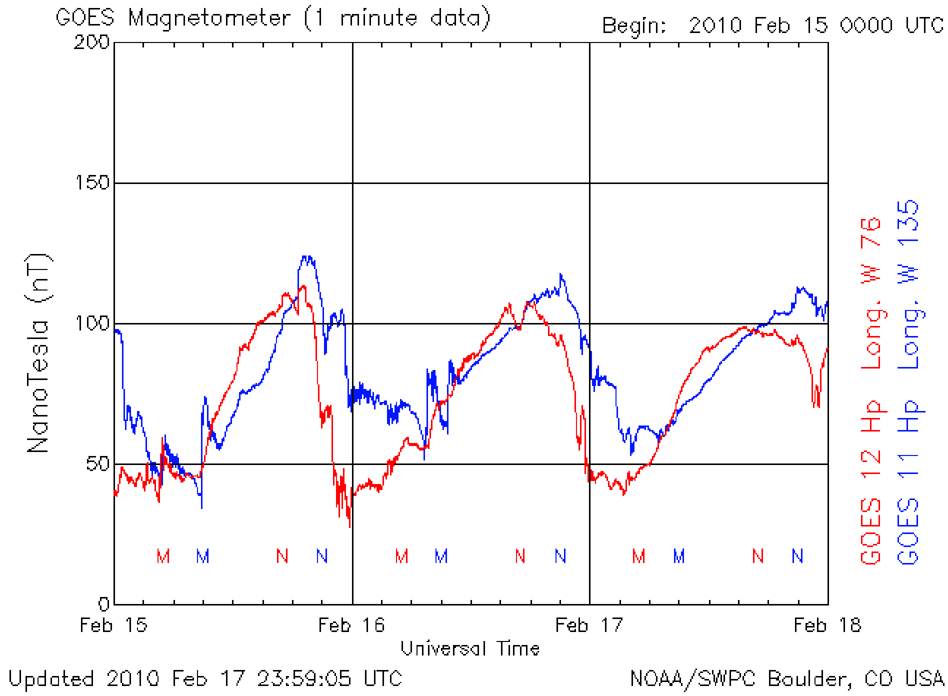

Figure 3.1: GOES magnetometer data for 15-18 February 2010. The CHARM-II launch occurred at 9:49 UT on the 16th, motivated by the preceding steep ~ 20 nT bay in the field.

a wave-particle correlator on an auroral rocket. The design was very similar to the instrument described below, except that the wave period was divided into only four phase bins rather than sixteen; the experiment revealed evidence of wave-particle interactions but could not resolve resistive versus reactive components. *Kletzing et al.* (2005) reports results from an instrument nearly identical to the one described below, launched into nighttime aurora. For various reasons, the experiment measured correlations associated with a relatively small number of the most intense Langmuir waves encountered, but the results gave a strong indication of wave trapping of the bunched electrons in those examples.

The success of previous wave-particle correlator experiments inspired a series of rocket experiments denoted the Correlation of High-Frequency and Auroral Roar Measurements (CHARM). The first CHARM launch experienced a payload system failure that precluded any correlator data-taking. A re-flight, CHARM-II, was launched from the Poker Flat Research Range near Fairbanks, AK, at 9:49 UT/22:46 MLT on 16 February 2010, reaching an apogee of 802 km. The launch, shown in Figure 3.2, was into an active substorm expansion phase, characterized by a 20 nT bay in the H-component of the magnetic field observed by GOES 11, as seen in Figure 3.1. The payload carried a Dartmouth High-Frequency Experiment and University of Iowa Correlator, as well as a number of other primary and contextual instruments. One particularly intense event encountered was reported on by *Kletzing et al.* (2011).

This work presents a comprehensive investigation of the entire wave-particle correlator data set from the CHARM-II mission. Section 3.2 covers the instruments which make up the

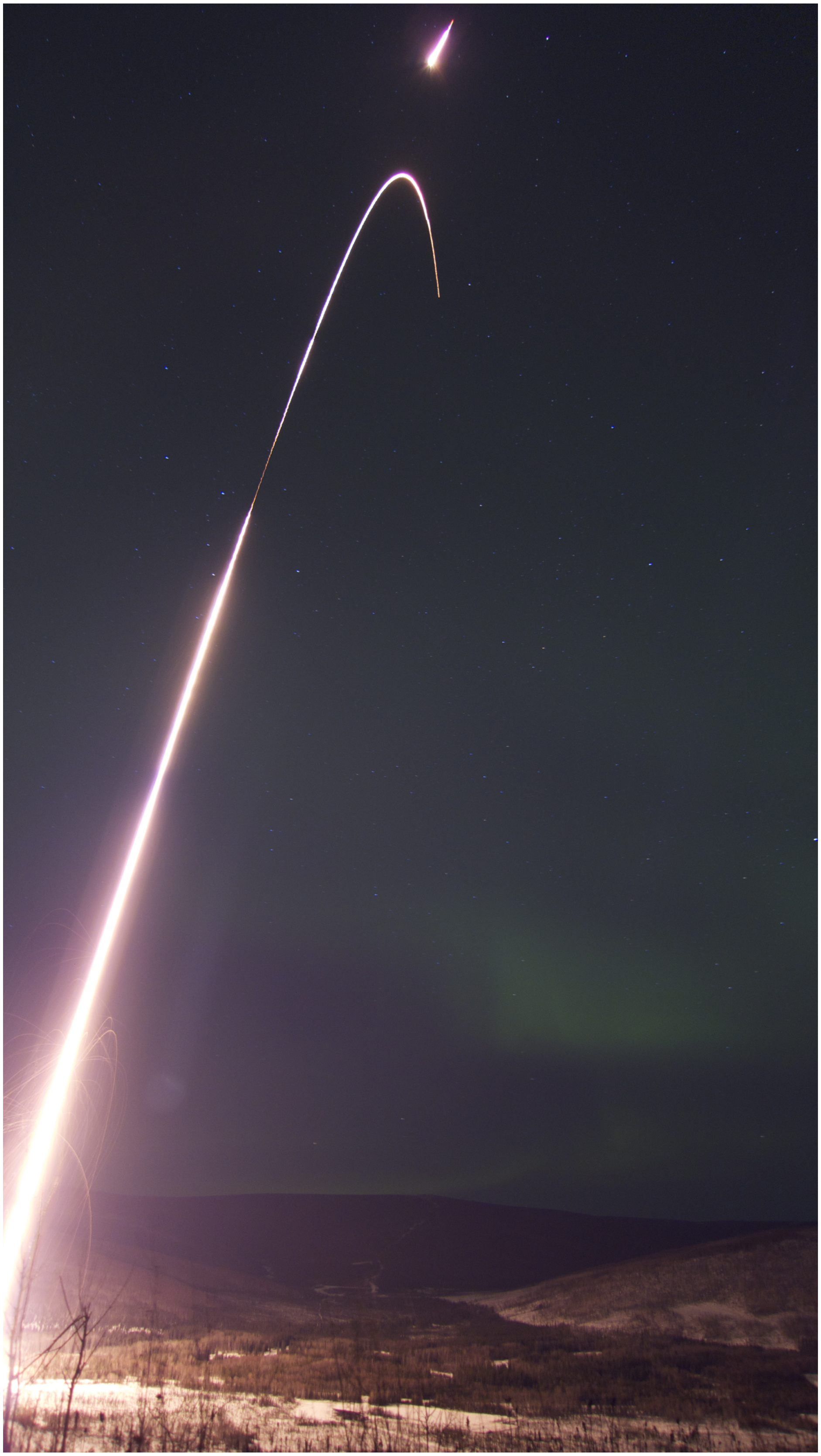

Figure 3.2: A photo of the CHARM-II launch.

Correlator system, and the form of the returned raw data. Section 3.3 presents the methods used to identify significant correlation events, a measure which characterizes individual events, several contrasting events from the final set, and observations regarding the set as whole. Section 3.4 summarizes these results and offers two theories developed to explain the observations. Finally, Section 3.5 develops a numerical test-particle simulation to test the plausibility of these theories.

3.2 Instrumentation

Accurate, in-situ correlation of Langmuir waves and electrons requires three primary pieces: a wave instrument covering the range of frequencies in which Langmuir waves are expected, high-speed particle detectors at a range of potentially resonant energies, and the correlation hardware itself, which processes these data streams, and reduces and returns the desired statistics.

The Dartmouth High-Frequency Experiment (HFE) detects the potential difference between two 2.5 cm spherical probes, separated by 30 cm along the payload's spin axis. This ΔV signal provides an estimate of the the axial component of the electric-field, which is mainly parallel to the ambient magnetic field, since the payload is kept field-aligned to within $\sim 10°$ by an attitude control (ACS) system. Active preamplifiers inside each spherical probe assure that the antenna functions as a double-probe over the entire 0-5 MHz frequency range. The signal is band-pass filtered to a 100 kHz to 5 MHz band, and regulated by an Automatic Gain Control (AGC) system to enhance the dynamic range. The AGC control signal is sampled onboard at 20 kHz and telemetered with other digital data. The regulated HF signal directly modulates a 5 MHz-bandwidth S-band transmitter, and the resulting waveform is continuously digitized at the ground telemetry station at 10 MHz, with 12-bit resolution. This instrument is the latest iteration of a design which has flown on numerous other rocket campaigns in both $\boldsymbol{E}_{\|}$ and $\boldsymbol{E}_{\perp}$ configurations, including HIBAR (*Samara et al.* 2004), PHAZE II (*McAdams et al.* 1998), SIERRA, RACE (*Samara and LaBelle* 2006), and ACES (*Kaeppler et al.* 2011).

The University of Iowa Wave-Particle Correlator similarly has heritage on numerous sounding rocket missions, including RACE and CHARM, and is described in detail by *Kletzing et al.* (2005). The Correlator takes an input waveform from the HFE, and uses it to control a phase-locked loop (PLL) circuit running at 16 times the frequency. The PLL phase-locks onto the frequency of the highest-amplitude component of the incoming wave, and restores to baseline, maintaining a 50% duty cycle. In the case of the HFE signal, the waveform is strongly dominated by the component at the Langmuir frequency when plasma waves are unstable. Under this condition, the PLL produces a clean, square-wave version of the Langmuir wave. This wave is then divided into 16 bins along its phase, and incoming counts from each of four detectors are sorted into these bins during an integration period—1 ms in the CHARM 2 case—corresponding to hundreds of wave periods per timeslice.

For the CHARM-II mission, two correlators were flown, each receiving particle data from four 'bagel' particle detectors. These detectors, named for the baked goods they resemble,

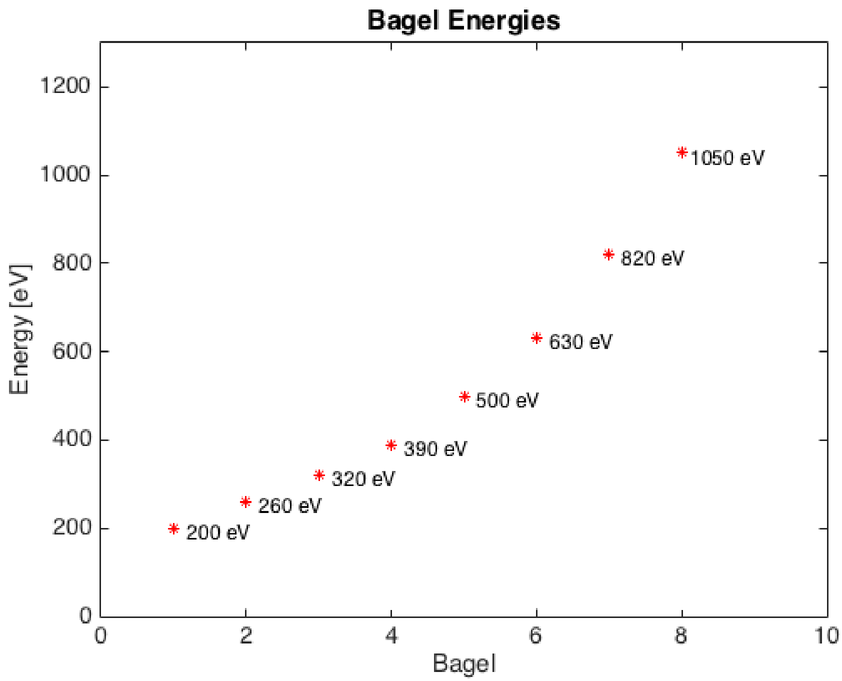

Figure 3.3: The energies of the eight 'bagel' particle detectors.

have energy acceptance ranges of 10%, and are characterized by a large geometric factor, as required for correlation with high-frequency waves (*Kletzing and Muschietti* 2006). The detectors are aligned with the rocket's spin axis, with a 10°-wide field of view, and thus are always observing the field-aligned component of incoming particles. The bagel detectors were tuned to logarithmically spaced energy levels ranging from 200 to 1050 eV (see Figure 3.3).

An additional contextual instrument of interest is the Electrostatic Electron Pitch Angle Analyzer (EEPAA), a 'top-hat' style detector which counts electrons, sorted into 15°-wide pitch angle bins and 47 logarithmically spaced energy bins from 15 eV to 15.5 keV, with a 50 ms integration time.

3.3 Data Presentation

Figure 3.4 shows a summary plot of the active period of the CHARM-II flight, with both time after launch and altitude on the x-axis. On top is a spectrogram of EEPAA data, with energy on the y-axis, and the log of differential flux as color intensity. Middle is an HFE spectrogram, with frequency on the y-axis, and color following wave power in decibels. Finally, the bottom is a plot of $\log_{10}$ of total counts among all eight Bagels. In the EEPAA data, an inverted-V structure is clear from approximately 610 to 660 s, with a more tenuous one from 500 to 560 s. The upper cutoff to noise which is near 500 kHz on the left of the HFE panel is interpreted as the Langmuir frequency ω_p, which acts as an upper bound to whistler modes in 'underdense' plasmas, where ω_p is less than the cyclotron frequency ω_c. From this,

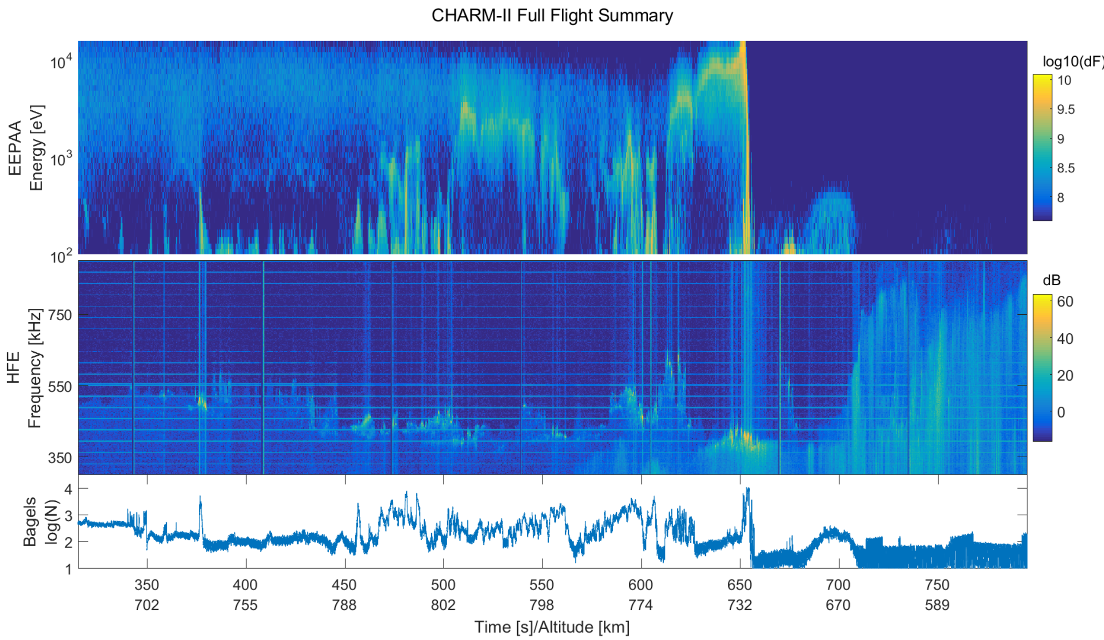

Figure 3.4: A summary plot of the active period of the CHARM-II flight. Both time after launch and altitude are shown on the x-axis. Top is a spectrogram of EEPAA particle data, with energy on the y-axis, and $\log_{10}$ of the differential flux [N * eV/(cm^2 * ster * s * eV)] as color intensity. Middle is an HFE spectrogram, with frequency on the y-axis, and color following wave power in decibels. Bottom is a plot of $\log_{10}$ of the total counts among all eight Bagels detectors. Note a clear inverted-V structure at 610 to 660 s, and a more tenuous one from 500 to 560 s. The upper cutoff to noise near 500 kHz on the left of the HFE panel is interpreted as the Langmuir frequency ω_p.

it is clear that the Langmuir frequency is much lower than the Upper-Hybrid Frequency ($\sim$ 1.4 MHz), and so easily selected for despite variance in the rocket's alignment parallel to the magnetic field. Given the bagel energy range it can also be deduced that the wavelengths in question will range from approximately 10 to 60 meters. Finally, it is clear that there are many instances where increased particle counts are accompanied by wave activity near ω_p. The activity near 650 to 660s was a particularly powerful event which saturated all onboard electric-field instruments at its peak, though the HFE data (and thus the Correlator) was only unusable for a few milliseconds. The peak Langmuir-wave electric-field intensity was estimated between 1 to 3 V/m, with the Langmuir frequency near 350 kHz.

The Correlator system returned approximately 489 seconds of valid raw data, providing a matrix with counts $s(t, p, E)$ at each of 488,869 timeslices (t), 16 phase bins ($p = 0...15$), and 8 energy levels ($E = 1...8$). While the phase between particles and the input waveform varies based on frequency, and due to daisy chaining of the HFE signal between the two Correlators, a first step taken to aid comparability of timeslices is to shift all bins to the same baseline, based on the recorded Correlator frequency. While the most direct way of looking at this data might be to display raw counts vs. phase and time, it is generally more edifying to

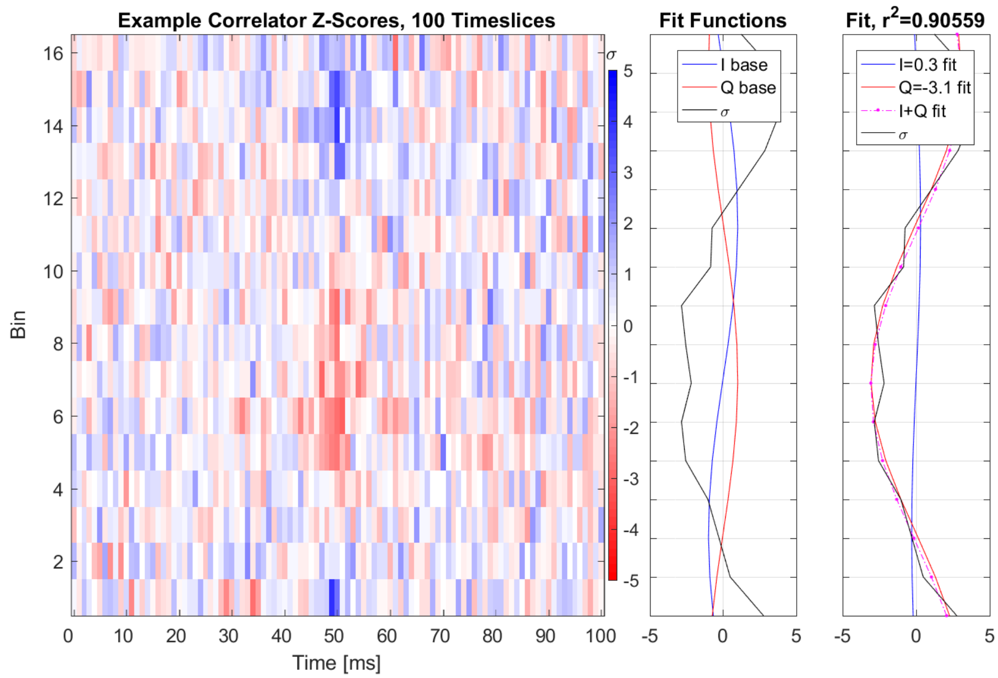

Figure 3.5: An example plot showing (left) the Poisson z-scores (σ) vs. phase and time for 101 ms of CHARM-II Correlator data. Z-score from -5 to 5 is shown as blue-to-red color scale, electric field phase is on the y-axis, and time (in relative ms) on the x-axis. Note that while a high-$|\sigma|$, multi-timeslice event is clearly visible in the middle timeslices, many more timeslices are insignificant. In the left line plot, the σ values for the central timeslice are shown, along with the base (unity amplitude) functions for an I/Q Resistive/Reactive fit, while in the right line plot they are shown with the fit amplitudes and their sum. The fit is reasonable, as shown by the r^2 value.

examine the Poisson z-score for the data,

$$\sigma(t,p,E) = \frac{s(t,p,E) - \bar{s}(t,E)}{\sqrt{\bar{s}(t,E)}},$$

where $\bar{s}$ is the mean particle count of the timeslice, $\bar{s} = \frac{1}{16}\Sigma_p s(t,p,E)$. This is a measure of particle over- or under-density in a given bin, with respect to the mean for that timeslice. This analysis yields plots as in Figure 3.5 (left), showing z-scores for 101 timeslices, with time on the x-axis, electric field wave phase on the y-axis (with each timeslice shifted such that the zero phases are aligned), and z-score shown as color.

While Figure 3.5 shows significant timeslices, it also shows large regions of low significance and noise. Likewise, perusal of the complete set of timeslices, $\mathbb{F}$, makes clear that many timeslices can be discarded due to a lack of significance and/or natural and instrumental interference, and indeed each timeslice can be classified by Correlator telemetry as ‘locked’ or not, and less than 15% of the data set has both Correlators locked. In addition, a timeslice cannot be considered reliable merely from the presence of a lock state at that timeslice, and this says nothing about the presence of interesting activity at that time. With so much data, manual inspection was not a practical or desirable method to identify reliable lock or significance, so an automated algorithm for event identification was developed.

The reduction analysis is motivated by an expectation of how significant wave-particle correlations will manifest themselves in the data: as a sine wave in the phase bins, with a quarter of the bins having a statistical excess of counts, and a quarter having a deficit. The pattern arises because of the bunching of the particles in the electric field of the Langmuir waves to which the PLL is locked.

The first step towards reduction of the Correlator data to identify discrete 'events' is, as shown in Figure 3.6, re-binning the 16 raw data phase bins p (orange) into four reduced phase bins p' (color coded by p'),

$$s'(t,p',E) = \sum_{n=0...3} s(t,4p'+n,E),$$

in order to emphasize the expected pattern of a quarter of the bins having over and under-dense counts. The re-binning was done four times ($q = 1...4$, depicted as individual columns of p' bins), shifted by one raw bin for each, to cover all possible patterns that might result from a wave-particle correlation event,

$$s'_q(t,p',E) = \sum_{n=0...3} s(t,4p'+n+q,E),$$

and the z-score,

$$\sigma'_q(t,p',E) = \frac{s'(t,p',E)-\bar{s}}{\sqrt{\bar{s}}},$$

was then calculated for each of these reduced timeslices.

To account for events which span multiple timeslices, two 'doublet' and one 'triplet' sets were constructed at each timeslice, acting as additional arrays of timeslices (the large divisions in Figure 3.6). For these, the means and the counts in each reduced phrase bin were integrated in time over the two or three raw timeslices, with the doublet sets defined as

$$s'_{q,t2b}(t,p',E) = \sum_{\tau=t-1,t} s'_q(t,p',E) \text{ and } s'_{q,t2f}(t,p',E) = \sum_{\tau=t,t+1} s'_q(t,p',E),$$

and the triplet

$$s'_{q,t3}(t,p',E) = \sum_{\tau=t-1,t,t+1} s'_q(t,p',E),$$

i.e. the doublets integrate either the timeslice prior or after, and the triplet both. All three sets then have their associated $\sigma'_{q,t2b}$, $\sigma'_{q,t2f}$, and $\sigma'_{q,t3}$. Thus, the process yields four total σ' arrays over the four q values, times four timeslice arrays (singlet, doublets, triplet), or sixteen total arrays.

Finally, as a criterion to identify timeslices with interesting events, we find the global minimum and maximum σ' over the sixteen arrays at each timeslice. The difference between

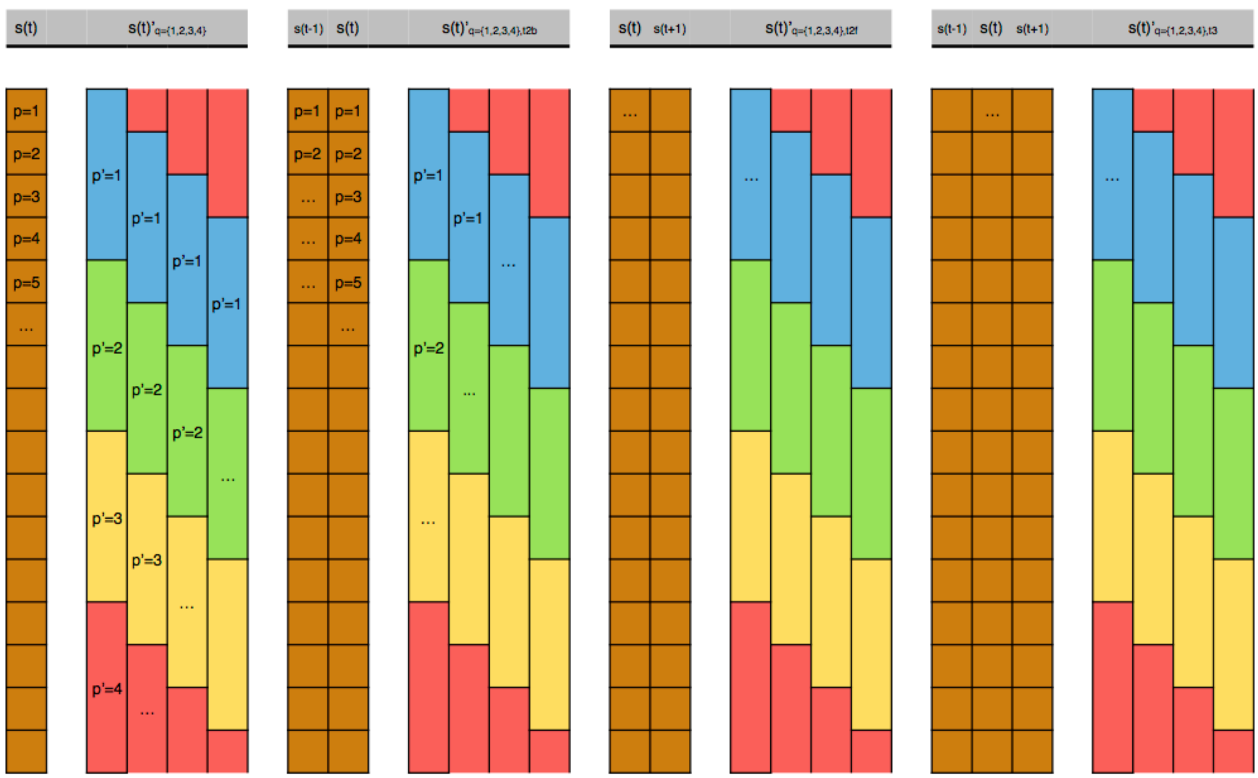

Figure 3.6: A map of the rebinning done at each timeslice to the raw correlator counts. Each large group is a time integration (singlet, doublets, triplet), and each of those contains four shifted rebinnings from the original 16 correlator bins of $s(t,p,E)$ (shown in orange), to the 4 bins of $s'_q(t,p',E)$ (shown in other colors, grouped by p').

that min and max, Δ, is then a scalar measure of how well a given timeslice matches the expected signature of a wave-particle correlation event.

Initially, a simple global threshold was used to find potential events, resulting in hundreds of identified timeslices; however, thorough investigation of these revealed many false positives among these sets, including many events which were disqualified after examination of diagnostic and contextual data. Figure 3.7 shows the two major factors which lead to disqualifying timeslices. In the second plot from the bottom, a powerful interfering signal can be seen in the raw HFE waveform data, at a cadence of about 15 μs. This signal, originating on the rocket payload, is frequently seen in the HFE data throughout the flight, and in some cases is the highest-amplitude component of the waveform. While the Correlator filters out such a low frequency, its presence is often correlated with the second disqualifying factor: a dubious recorded PLL lock frequency. Visible as the solid white line overlaying the HFE spectrogram in the bottom plot, the lock frequency can either be unstable (i.e. not properly locked), or set to unrealistically high or low values compared to by-eye evaluation of the Langmuir cutoff (dashed black overlaying line). In either case, this drastically reduces confidence in the correlator phase binning correctness, and motivates the discarding of such events.

The large number of false identifications in the initial run also revealed that the Δ-threshold needs to be different for each bagel detector. With these issues in mind, the event identification algorithm was altered to iteratively optimize the Δ threshold for each bagel. Subsequently, manual screening was applied based on the considerations above. The 820 and 1050 eV bagels had no qualifying timeslices. Figure 3.8 shows an overview of the final set $\mathbb{S}$ of thresholded, hand-screened events, as x-marks at a given bagel and time, overlaid on a 20-bin histogram. From the histogram it is clear that majority of the events are in a tight cluster near 652 seconds, coincident with the strong event at the end of the major inverted-V structure seen in Figure 3.4. Two longer clusters of events are centered near 490 and 610 seconds. The per-bagel set sizes are shown in the table inset in the upper left of Figure 3.8, showing that the majority of the events were at 260 and 630 eV, with 12 and 23 events, respectively.

Linear analysis of the interactions of Langmuir wave packets with electrons has shown that the perturbation of the electron distribution function can be broken into two components: the 'resistive' component of trapped particles which oscillate in-phase with the electric field of the Langmuir wave, or 180 degrees out of phase, and the 'reactive' component which oscillate 90 or 270 degrees out of phase (*Kletzing and Muschietti* 2006). A strong resistive component is an indicator of wave-particle energy exchange, leading to wave growth or damping, while the reactive phase is associated with particle trapping. The summation of these two components will tend to have a sinusoidal form when either component shows significant activity, and it is this form that the event-identification method focused on.

To look at the Correlator data in a comparable manner, we fit the correlator timeslices to a quadrature function vs. bin number p,

$$-I\sin\left((p-p_0)*\frac{\pi}{8}\right)-Q\cos\left((p-p_0)*\frac{\pi}{8}\right)$$

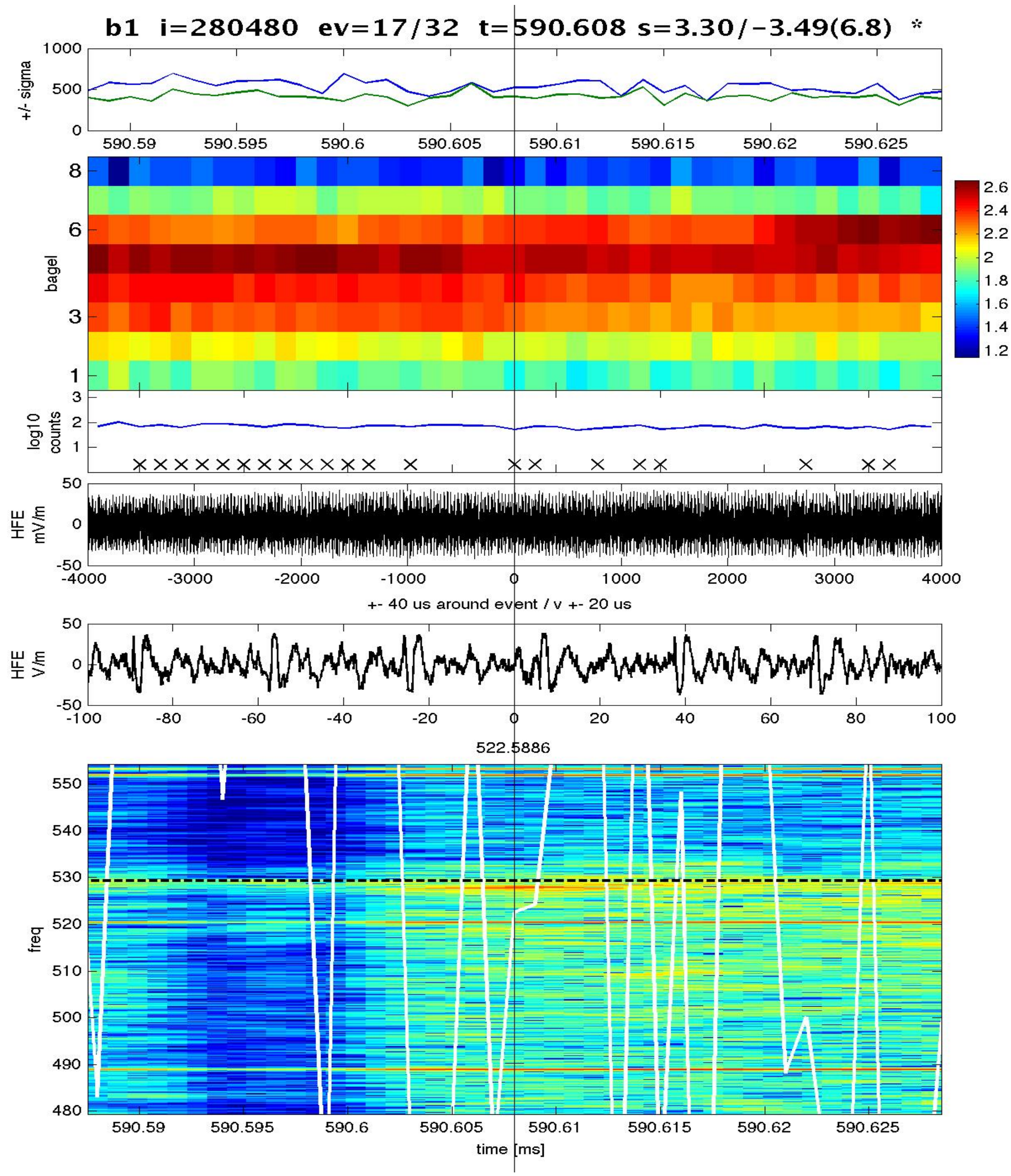

Figure 3.7: An example of plots used to hand-screen events. From the top, the plots show max(σ) and min(σ), a counts vs bagel & time spectrogram, the logarithm of raw counts for a single bagel (with PLL lock/no-lock status displayed at the bottom, as an X for 'lock'), two timescales of raw HFE waveforms, and an HFE spectrogram with PLL lock frequency (white, solid) and hand-picked Langmuir frequency (black, dashed) overlaid. This event was discarded because it shows both a wildly fluctuating lock frequency with only sporadic 'lock', and strong periodic interference of unknown origin at approximately 15 μs cadence.

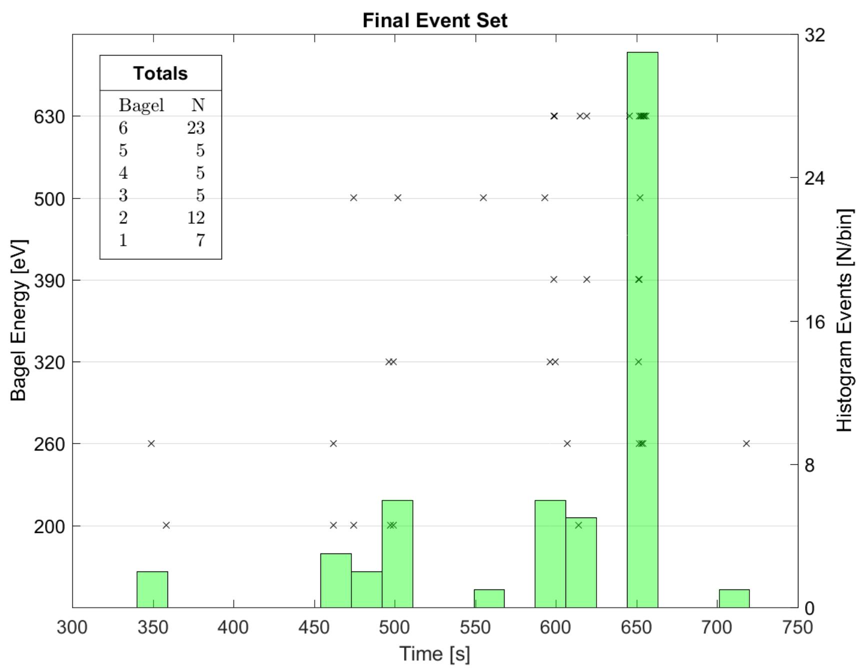

Figure 3.8: An overview of the final event set $\mathbb{S}$ vs. time for the whole flight, with the per-bagel totals in the table inset top left. Individual events are displayed as x-marks on lines corresponding to bagels on the left vertical axis. Overlaid on this is a histogram of events vs. time, corresponding to event/bin counts on the right vertical axis.

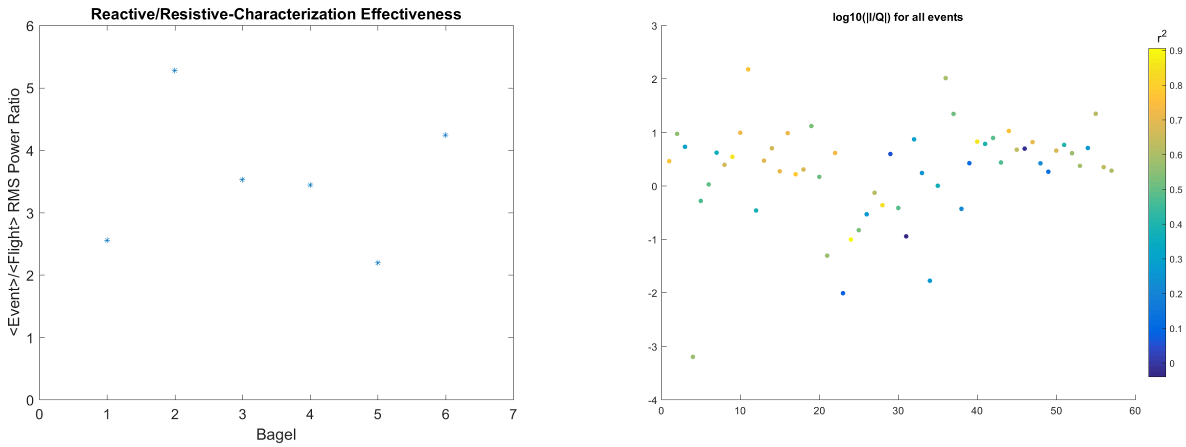

Figure 3.9: On the left, for each bagel, the ratio of $\langle\sqrt{I^2+Q^2}\rangle_{\mathbb{S}}/\langle\sqrt{I^2+Q^2}\rangle_{\mathbb{F}}$, showing that the events in $\mathbb{S}$ show significantly more reactive/resistive activity than seen in the general flight. On the right, a plot showing the resistive/reactive power ratios, as $\log_{10}|I/Q|$, for all of $\mathbb{S}$, showing some variance, and resistive activity in a majority of events.

optimizing I and Q to best fit the z-scores. The negative signs and p_0 are determined by calibration data relating the electric-field phase to the bins. After performing this type of fit on all timeslices, the coefficients I and Q are then magnitudes of the In-phase and Quadrature signals, i.e. the resistive and reactive linear components. An example set of fits are shown in the line plots on the right of Figure 3.5, with bins on the y-axis (aligned to the left σ plot)—in both line plots, the black line is the σ values for the center timeslice. In the left plot, the solid blue line is the calibrated electric field, corresponding to I, the resistive component, and the dashed red line is reactive Q component. The right plot shows fitted forms, with blue and red the fitted, separate I and Q, and the dot-dashed magenta line their sum (i.e. the actual fit function). For this timeslice, the r^2 goodness-of-fit is ~ 0.906, showing a reasonable fit, and with $I \sim 0.3$ and $Q \sim -3.1$ this event appears to be dominated by the reactive component.

Figure 3.9 shows diagnostics of this fitting. The plot on the left has the per-bagel ratios of the mean magnitudes I and Q when taken over only $\mathbb{S}$, compared to the mean magnitude for all of $\mathbb{F}$. From this it's clear that this form of analysis does a reasonable job of characterizing events, though it is worth noting that it yields enough false positives and negatives to make it less useful than the above-outlined method for event identification. The right plot shows the log of the magnitude of the resistive-to-reactive power ratio, $\log_{10}|I/Q|$, for $\mathbb{S}$, revealing significant variance, and that a majority of events are more resistive.

Figure 3.10 displays several events of interest from $\mathbb{S}$. Each shows, from the top, 1 second of contextual data from the EEPAA, a spectrogram of the total counts of each bagel vs. bagel energy, the values of I, Q, and $\sqrt{I^2+Q^2}$, and σ vs. wave phase.

Considering the top left stack of Figure 3.10, which shows data from an event identified in the 260 eV bagel's data, we see some evidence of higher-energy beams in the EEPAA data leading up to the event, but nothing significant during it. The bagel spectrogram shows

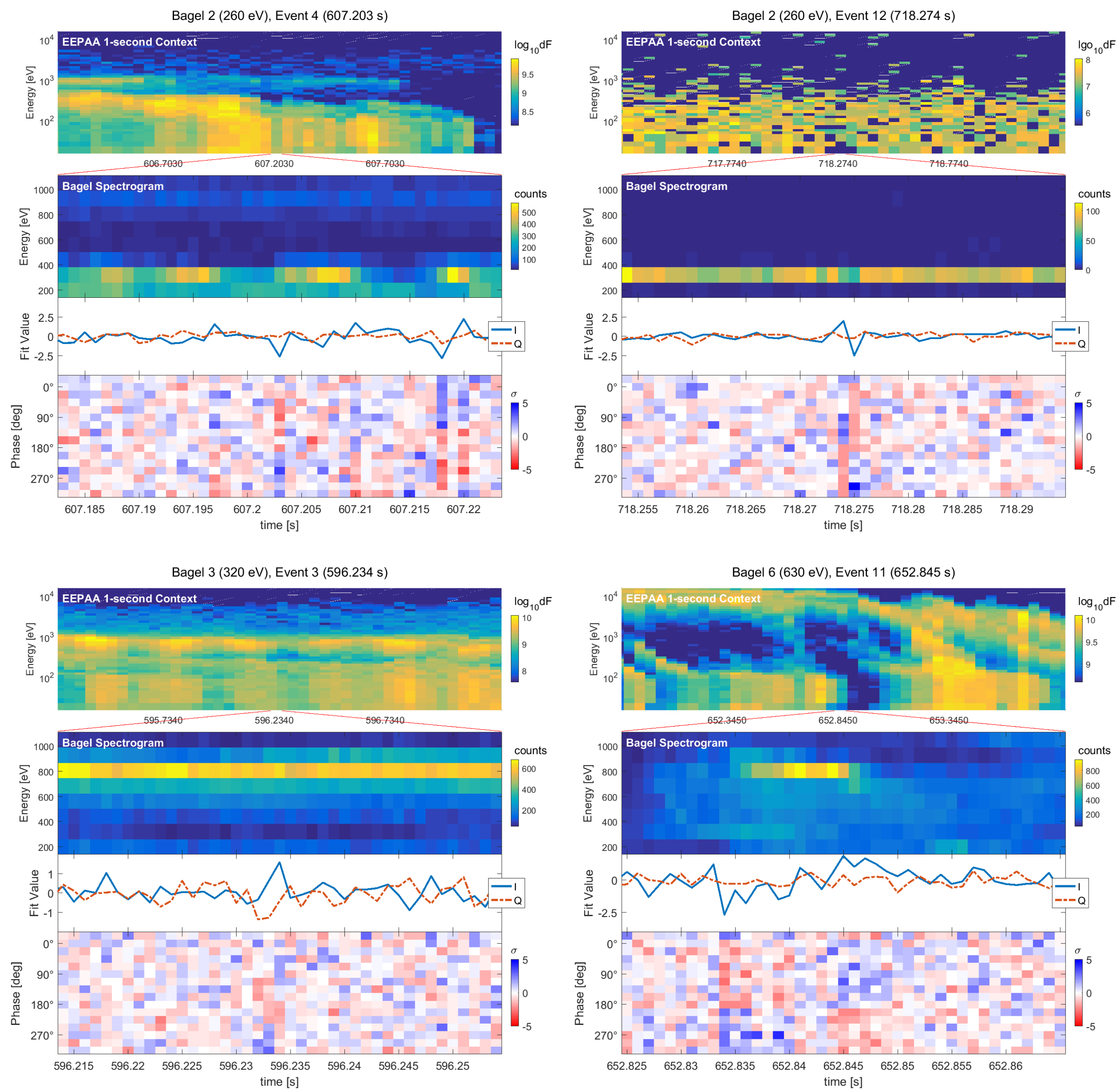

Figure 3.10: Four example events from the final, Δ-thresholded, hand-screened set $\mathbb{S}$, with 1 second of EEPAA/top-hat particle detector data on top, for context, followed by a spectrogram of bagel total counts vs. energy, a line plot of reactive/resistive fit values, and a $\boldsymbol{\sigma}$ vs. wave phase plot. Note the presence of multi-timeslice events in the $\boldsymbol{\sigma}$ data, as well as short-lived narrow-band beam features in the bagel spectrogram, the presence of different 'red over blue' and 'blue over red' regions in close temporal proximity, and the behavior of I and Q during these events.

sporadic, short-timescale (2-5 ms) beams in this bagel alone. The central timeslice in the σ plot is the event which passed the thresholding and hand-screening, identifiable as a vertical bar with strong blue bins in the top half, and strong red bins in the bottom half, showing the characteristics which were selected for by the thresholding process. There are several other timeslices around this event which show similar signals, or signals with red bins over blue bins, and it is clear that the resistive component of our I/Q fit is differentiating between the red-over-blue and blue-over-red cases. It also appears that the stronger beams show a relation to I, with negative-I events preceding beams, and positive-I following.

In the other plots of Figure 3.10, we see that while there are several cases of timewise-longer events, the majority are single-timeslice (1 ms) events as in the top left, a trend that holds throughout $\mathbb{S}$. The top-right stack shows extremely low counts on the EEPAA for unknown reasons. While only the lower-right stack shows a clearly distinct beam, the pattern does appear to hold that a negative I comes before or during a density increase, while a positive I correlates with a density decrease.

Figure 3.11 highlights the event previously presented by *Kletzing et al.* (2011), which covers a larger time and energy range. This figure shows the same parameters as those in Figure 3.10, but I/Q and σ plots are shown for the 200 to 630 eV bagels. The EEPAA data shows clear evidence of a dispersive beam appearing at or slightly before this event, lasting for approximately 200 ms, and the bagel spectrogram and σ plots also appear to show this at finer timescales. For this event, both I and Q show strong responses, and I appears to show the same association with the density gradient at a given bagel.

Finally, Figure 3.12 shows a set of events which take place over a similar time scale to those in Figure 3.11. Evidence for dispersion is less clear here, especially given the large gap between the 260 eV and 630 eV bagels in which events are visible. However, in each of these examples I displays the same relation to the beam as observed in Figures 3.10 and 3.11; that is, that a negative I is associated with a density increase, and vice versa.

Figure 3.13 shows scatter plots of I and Q values versus two selected parameters, the temporal gradient of the electron beam flux ∇n_B, inferred from the bagel detectors (upper row), and the temporal gradient in the RMS HFE waveform $\nabla < w >$ (lower row). The points are colored according to their r^2 goodness-of-fit value from the reactive/resistive fitting. The upper left panel demonstrates from this statistical approach the correlation between the I value and the beam gradient which has been illustrated by multiple examples in Figures 3.10 and 3.12. A clear trend is evident whereby negative I values correspond to positive beam flux gradients, and vice versa. A linear regression to these points returns a t-statistic value of -6.97 with a p-value of 4×10^{-9}. The other panels of Figure 3.13 show that there is no pattern evident between the Q value and ∇n_B, or between the I or Q values and $\nabla < w >$. Linear regressions of these sets all have small t-statistics, and p-values ≥ 0.15, strongly suggesting that the I-∇n_B relationship is the only significant one of those examined. Table 3.1 summarizes additional statistical tests performed on the data, showing that a Kolmogorov-Smirnov test finds a significant difference between the $\nabla n_B < 0$ and $\nabla n_B > 0$ distributions of I, and that I and ∇n_B are the only significantly correlated measures.

Additional by-eye comparisons were made between the I/Q responses and high-cadence

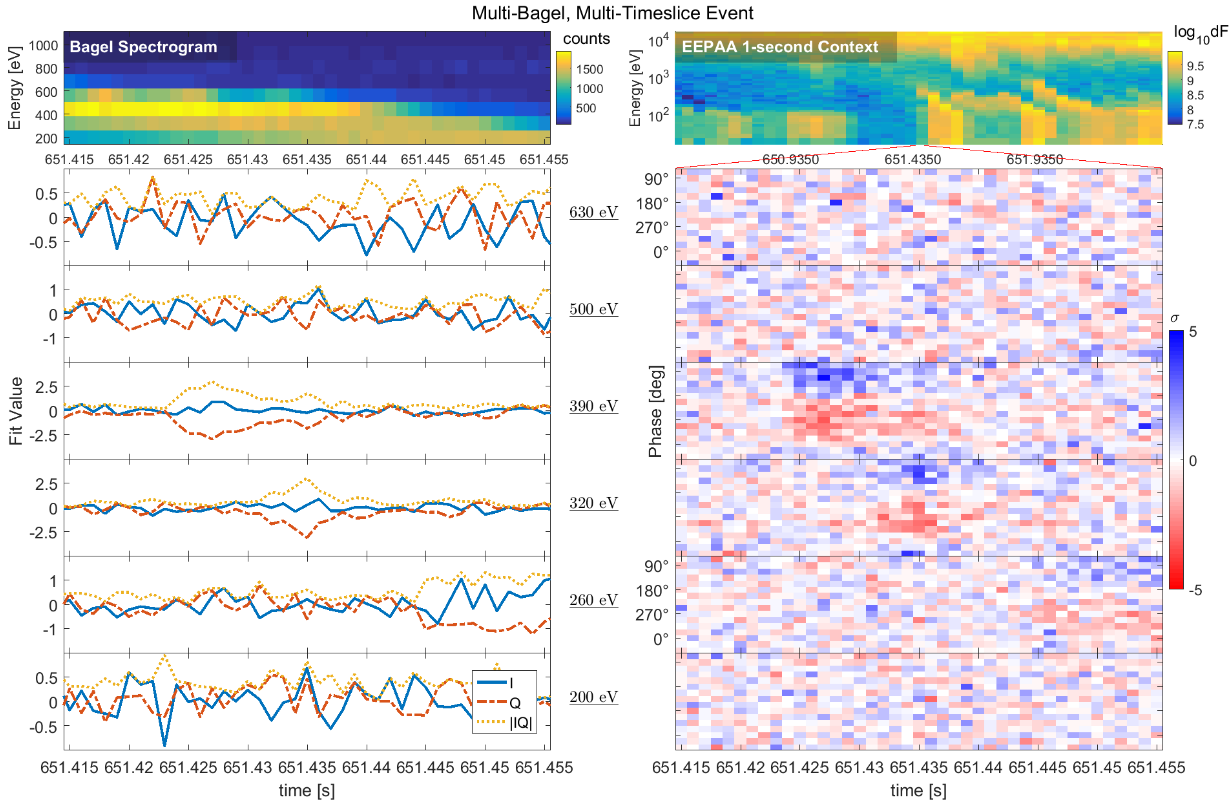

Figure 3.11: A single-event summary, showing bagel spectrogram (top left), EEPAA/top hat particle detector context (top right), and I/Q values (left column) and z-scores (right column) for each bagel. Note clear evidence of a dispersive beam passing through the 390 to 260 eV energy range, and the strong response in I and Q.

Table 3.1

Statistical tests on the Figure 3.13 scatter plots.

	I-∇n_B		Q-∇n_B		I-$\nabla{<}w{>}$		Q-$\nabla{<}w{>}$	
	s	p	s	p	s	p	s	p
Lin Regression	-6.97	4.0×10^{-9}	2.26	0.027	-0.36	0.72	1.27	0.21
K-S Test	1	2.8×10^{-8}	0	0.23	0	0.54	0	0.36
X-Correlation	-0.68	7.8×10^{-9}	0.24	0.062	-0.05	0.69	0.19	0.15

These statistics all evaluate relations from I and Q to ∇n_B and $\nabla < w >$. The ‘s’ heading is general, referring to the significant output of the given test: from top to bottom, the t-statistic, null-hypothesis rejection, and correlation coefficient.

In all tests, note the extremely low p-value of the I-∇n_B relation, compared to the others.

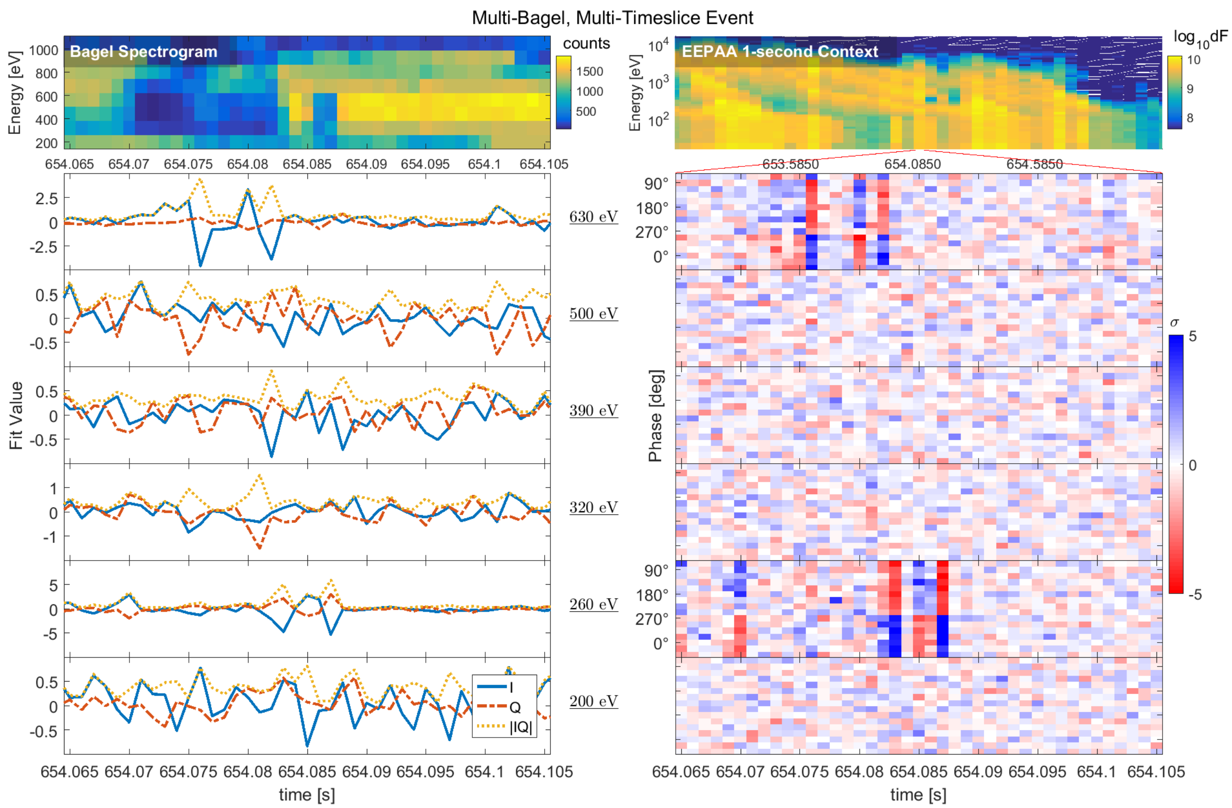

Figure 3.12: A single-event summary, bagel spectrogram (top left), EEPAA/top hat particle detector context (top right), and I/Q values (left column) and z-scores (right column) for each bagel. Note that while the EEPAA shows evidence of larger-timescale dispersive beams at higher energies, the same pattern does not seem to hold at the bagel energies.

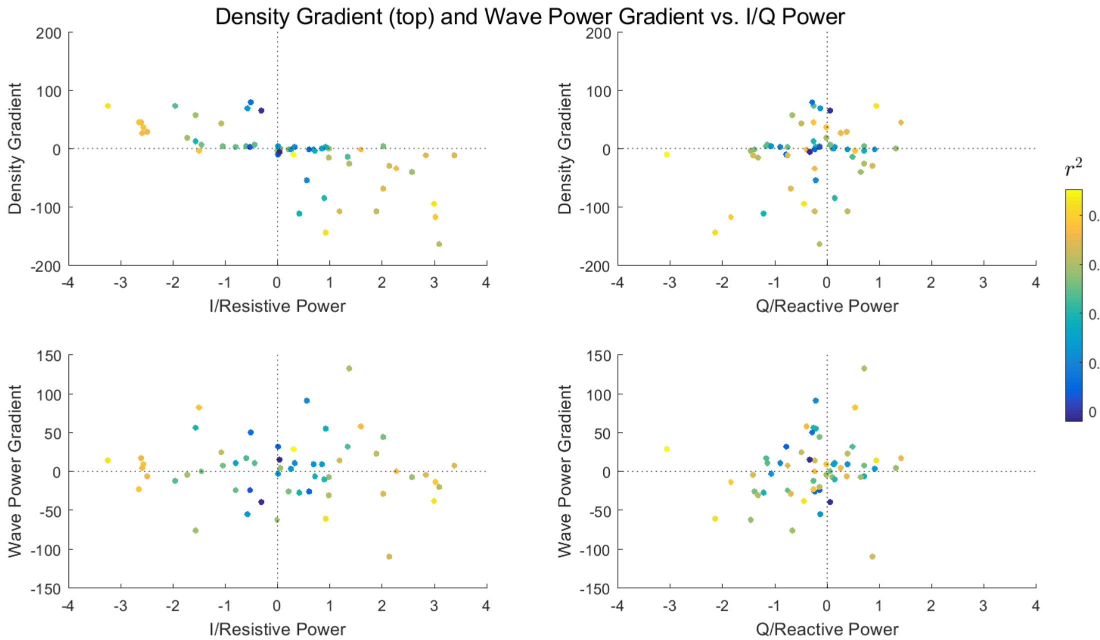

Figure 3.13: Scatter plots of bagel count gradients (top row) and HFE wave power gradients vs. I (reactive, left column) and Q (resistive) fit values, with color as the r^2 goodness-of-fit value. Note the clear relations in the I-to-count gradient (i.e. resonant electron density gradient) plot, and the lack of any relation in the wave power plots.

payload data streams. In particular, the Langmuir frequency—as judged by finding the whistler-mode cutoff on HFE spectrograms—was closely examined, but showed no obvious short-timeframe reactions in relation to I or Q changes. The I-∇n_B relation to-date remains the only relation seen.

3.4 Discussion

The above shows that the Correlator system has observed 57 potential wave-particle correlation events, after thresholding for significance and hand-screening to remove interference and potentially bad timeslices. An analysis of the reactive and resistive components of the Correlator event timeslices reveals a relation between a positive ∇n_B at a given bagel energy level, as in the case of an electron beam appearing at that energy level, and negative values of I for coincident Correlator events, and a similar relation between a negative ∇n_B and positive I. Given the calibration of the Correlators, the positive half of the electric-field waveform corresponds to a field pointing towards the bagels, and thus electrons being accelerated away from them. Thus, the observed relation is consistent with energy going from the beam to the wave field during a beam density increase, and the inverse for a density decrease. The lack of evidence for a relation between the HFE power and I and Q is curious, given prior observations by *Kletzing et al.* (2005) of such a relation on the RACE mission. It is possible that the extreme wave power during the majority of the events in $\mathbb{S}$ may mask such an effect, particularly given that the amplitude modulation typical of bursty Langmuir waves is also not prevalent in the CHARM-II HFE data.

There are at least two relatively simple explanations for the pattern of correlations observed between in-phase wave particle correlation phase and the gradient in the beam flux. The first is motivated by the clear evidence for a dispersive beam in the event illustrated in Figure 3.11. In fact, this type of dispersive beam is the normal pattern for parallel electron beams in so-called Alfvénic aurora, in which the beams are accelerated by Alfvén waves at altitudes well above the rocket, and undergo dispersion as they propagate to lower altitudes, with fast electrons outrunning slower ones (*Kletzing and Hu* 2001; *Chen et al.* 2005). Figure 3.14 shows a schematic of the situation, as such a dispersed beam passes by a low-altitude rocket payload. The beam energy decreases with time from left to right, and as shown, the energy range of an appropriate fixed-energy particle detector will shift from lying below the peak energy of the beam to lying above the peak energy. In the former case the detected energy corresponds to the condition $df/dv_{\|} > 0$, which is destabilizing for Langmuir waves, and the latter case corresponds to $df/dv_{\|} < 0$ which is stabilizing. Under the former condition one expects waves resonant to the detector's energy to be growing, extracting energy from the beam, which would correspond to the negative values of the in-phase component of the electron-electric field correlation. Under the latter condition, the opposite energy flow would be expected, corresponding to wave damping at the detector energy. The expected signature in the phase of the electron bunching is exactly as observed.

The Figure 3.12 event is a case where the explanation given above falls short. While there is evidence in the EEPAA data of dispersion at higher energies, the alternating patterns in the $\boldsymbol{\sigma}$ data are difficult to explain, as is the large energy gap between the two bagels which show a signal. An additional theory does not exclude effects from the above behavior, but rather focuses on the beam appearance and disappearance at a small region of energy and pitch-angle space. The two populations involved in this theory are the 'warm' background electrons which are a degraded secondary population associated with a beam, and the beam population itself. Depicted in Figure 3.15, the bagels observe a slice of the incoming particle distribution along the parallel axis—as a beam appears, the highest-energy particles will

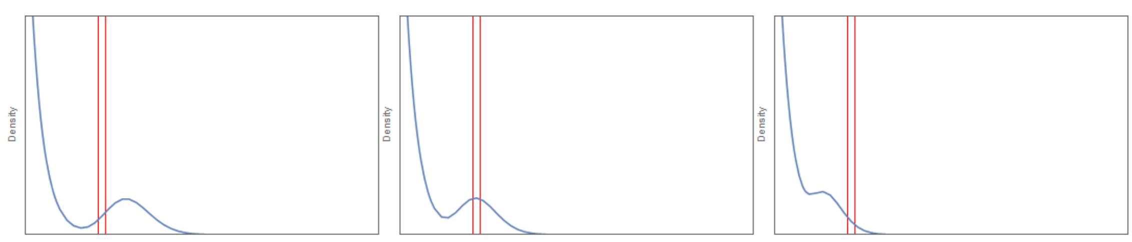

Figure 3.14: A cartoon showing the 'dispersive beam' explanation for the relation seen between I and Q and the particle count gradient. As time passes (left to right), the bagel detector's first sees the positive-slope region of the Maxwellian beam distribution, and then later a negative-slope region. Thus a beam passing downward through a bagel's energy range will see resonant wave growth, followed by enhanced damping.

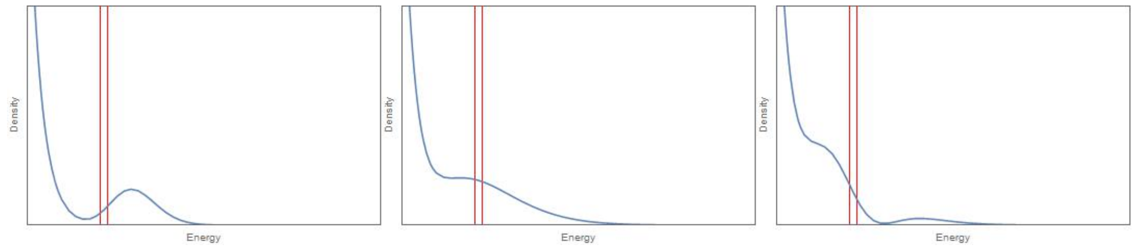

Figure 3.15: A cartoon showing a more in-depth explanation for the relation seen between I and ∇n_e. As a beam appears, the high-energy particles are the earliest to arrive, leading to an exaggerated positive slope and wave growth (left). The remainder of the particles and distribution relaxation then yield a plateau in middle times. Finally, when the beam turns off at the source, the high-energy particles are the first to disappear, and in the right configuration may yield an exaggerated negative slope, enhancing wave damping (right).

arrive first, and are likely to create a positive-slope region and wave growth. However, as the beam turns off, it is also possible, depending on the energies of the two populations, that an enhanced negative slope will appear as the higher-energy particles disappear first. This enhanced negative slope can then lead to enhanced damping in a narrow energy region. The presence of short-lived beam features in the top-left and bottom-right plots of Figure 3.10, and their temporal relation to the nearby correlator events, is compelling evidence supporting this theory.

Langmuir wave growth during an increase in the number of electrons at or near the resonant energy is generally expected because of the resultant instability, whether due to a beam moving into an energy range, or simply appearing at that energy. While subsequent damping is also expected, an impulsive enhancement of damping concurrent with the beam's disappearance, is, on the other hand, not an immediately obvious causal relationship.

Finally, in a high field, it is expected that unstable distributions will relax extremely swiftly, leaving mostly trapped, reactive particles to be observed; thus, the degree of resistive activity evident in the ratios on the right of Figure 3.9 is unexpected. These observations, and the fact that many of the events are narrow in time, many in a single timeslice, may suggest further structure at even shorter timescales.

A numerical analysis can confirm interpretations of the observed Langmuir wave growth and damping—and hence the phase of the in-phase wave-particle correlation for both positive and negative temporal gradients in the electron beam flux. It could also potentially allow probing of activity at shorter timescales. Towards this end, a method was developed to simulate the evolution of the electron distribution function, and thus the reduced distribution function and Langmuir wave growth rate, as an electron beam propagates through a vertical distance of approximately 5000 km in a converging magnetic field, with parameters close to those of the auroral ionosphere.

3.5 Simulation

The aim is to simulate a minimal-complexity environment that is sufficient to probe the questions at hand: does an electron beam with reasonable characteristics, and which shows significant Langmuir wave growth upon its appearance, also show enhanced wave damping as it fades? Can we say anything about the short-timescale behavior of the wave growth rate?

We shall define a 'reasonable beam' as originating with a Maxwellian distribution at a realistic altitude, and traveling through a magnetic field with an Earth-like mirror ratio. A similar analysis is performed by *Arnoldy et al.* (1999), and simpler methods such as one using a guiding-center approximation might suffice. However, interesting effects may be evident in this case and in broader applications, if detailed environmental attributes are taken into account, such as electric fields and agryotropic distributions. We therefore choose to develop a complex, flexible—and computationally intensive—test particle simulation system. Its development on and application to this case shall use simple, gryotropic magneto-kinetic parameters, with no inter-particle interaction or wave-particle scattering.

Following a numerical analog to the analytical method of *Cairns* (1987), we note Liouville's equation governs the evolution of a distribution function over time, and with no wave-particle or inter-particle scattering, we can simply write

$$f(\bar{x}, \bar{v}, t) = f(\bar{x}', \bar{v}', t'),$$

i.e. that the value of the distribution function at a source phase-space region $(\bar{x}', \bar{v}')$ at time t' is the same for the related region $(\bar{x}, \bar{v})$ at time t. The test-particle simulation is used to relate the primed and unprimed regions, by creating a lookup table of particle travel times $\mathbb{T}(\mathsf{E}, \alpha')$ for a range of source energies E and pitch angles α. These test particles are then treated as centers of regions in phase space, and are used to 'carry', in time $\mathbb{T}$, values of the source distribution function down to a corresponding region (E, α) at the observation point.

In this analysis, z is taken to be positive, downward, field-aligned coordinate, with $z = 0$ corresponding to the beam generation altitude. In order to only simulate particles which will arrive at our 'detection point' at $(x = y = 0, z = -5000$ km$)$ we use a deterministic (i.e. time-reversible) simulation method, and originate our test particles at the detection point with an upward velocity, watching for them to cross a target plane at $z = 0$. The velocities can then be reversed for the later downgoing analysis. The 'Boris Method' is used—a standard, time-reversible particle pusher (*Boris* 1970; *Birdsall and Langdon* 2005). This method separates the effects of the electric and magnetic forces, dividing them into a half-impulse from any background electric field, followed by a rotation according to the magnetic field, and then another electric half-impulse.

Careful testing of energy conservation led to setting a unitless timestep of 0.01. The base of the time system is the electron cyclotron frequency, and so this is equivalent to each timestep moving each particle a hundredth of an orbit. For the input parameters used, this yielded a worst-case energy loss of 0.06% over the full length of the simulation.

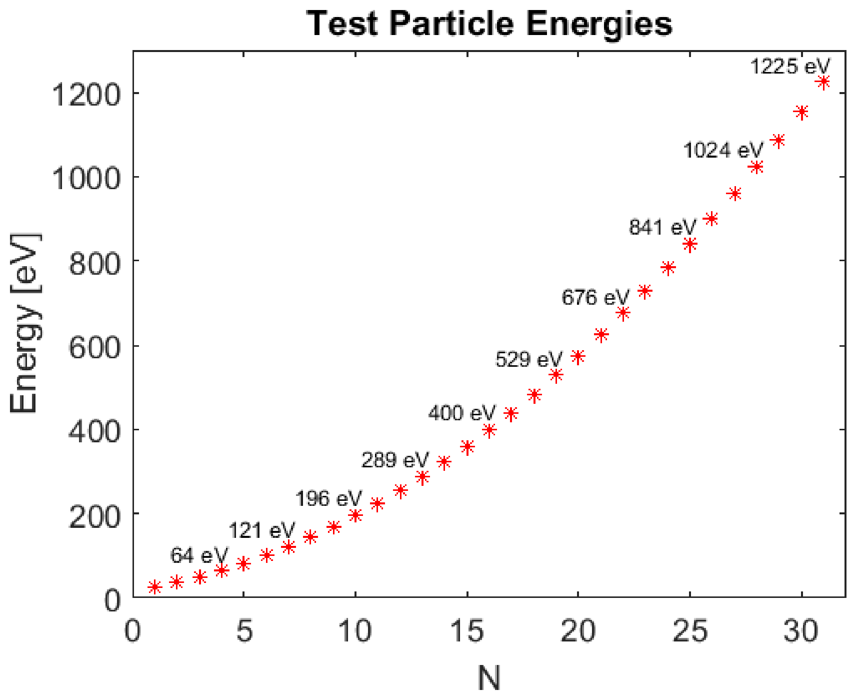

Figure 3.16: The 31 test-particle simulation launch-energy levels. Linearly spaced in velocity.

To allow a realistic amount of time/space for mixing of particles of different energies and pitch angles, a distance of 5000 km is used, corresponding to the distance from the bottom of the electron acceleration region to the ionospheric detection point. The background electric field is assumed zero, and the magnetic field is rotationally symmetric around $x = y = 0$, defined as

$$\bar{B} = -\frac{zr}{L^2}\hat{r} + \frac{1+z^2}{L^2}\hat{z} = -\frac{xz}{L^2}\hat{x} - \frac{yz}{L^2}\hat{y} + \frac{1+z^2}{L^2}\hat{z},$$

with L a scaling variable determined by our desired mirror ratio and target distance. For the following simulation, the mirror ratio was set to 5.

In order to fully cover the range of energies detected by the Correlator, particles were launched at the 31 energies plotted in Figure 3.16, linearly spaced in velocity, with energies ranging from 25 to 1225 eV. At each energy, particles were launched in a range of 41 pitch angles, with an angular spacing of $3\pi/256$. Because of the rotational symmetry of the simulation imposed by gyromotion, it is only necessary to launch particles at one azimuthal angle—the results can then be rotated to fill a velocity-space hemisphere at the detection point. As we would like our hemisphere segments to sweep out constant solid angles, we require that, for pitch angle α, the particle rotates to the integer number of azimuthal angles ϕ which most closely yield $\Delta\phi = \frac{\Delta\Omega}{\sin(\alpha)\Delta\alpha}$, where $\Delta\Omega$ is the solid angle, herein set to 0.001 steradians.

The simulation code was implemented in MATLAB, manually fragmented into 64 shards, and run in approximately five weeks on the Dartmouth Discovery cluster. Due to the finite

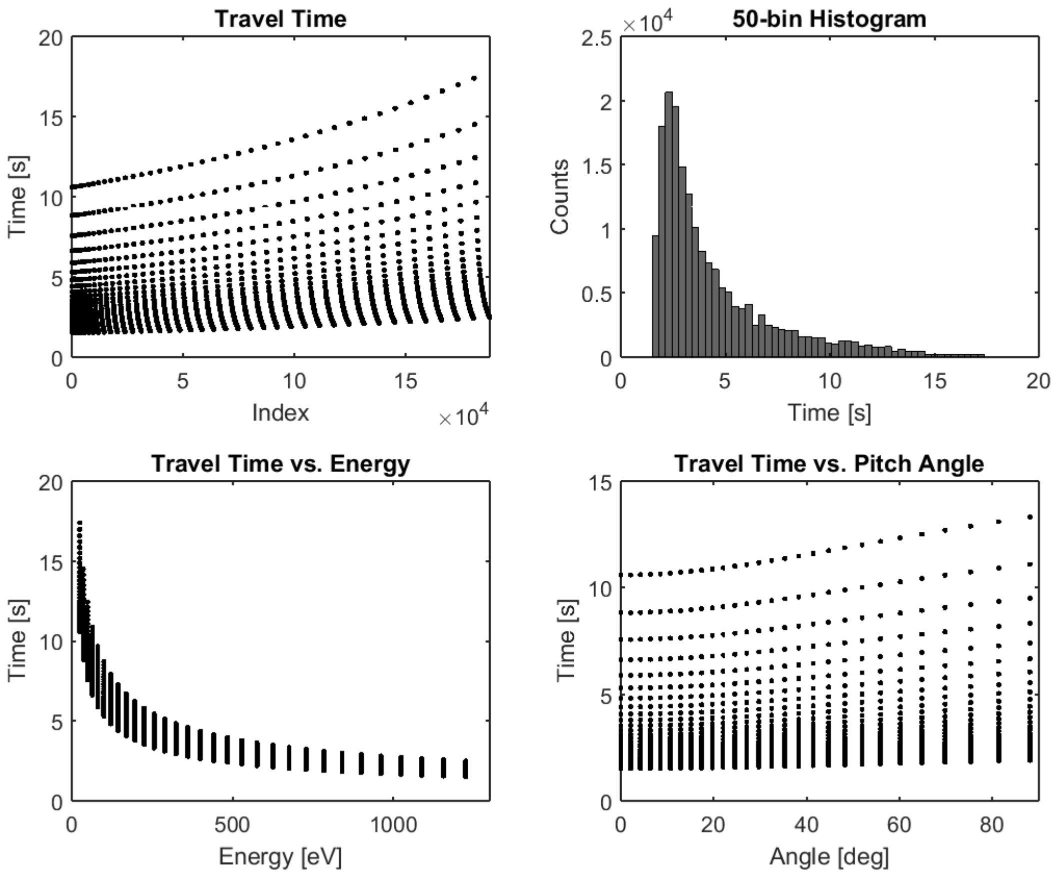

Figure 3.17: Various diagnostics of the resultant travel times for the simulated test-particles. Note that the energy and pitch angle in the lower row are the launch values at the detection point (high magnetic field).

timesteps, which are unlikely to land precisely on $z = 0$, interpolation was required to find exact crossing parameters. To enable this, the final 1000 timesteps for each particle were saved (with the very last step having $z > 0$), and gyro-orbit equations were fit to these, from which accurate final $z = 0$ position, velocity, and travel times come forthwith. All unitless values were then interpreted via inter-defined base values: $B_0 = 50$ microtesla, $t_0 = \sim 714$ ns, $r_0 = \sim 0.337$ m, and $v_0 = 0.00989$ c, corresponding to 25 eV.

Figure 3.17 shows some basic diagnostics of the output of this simulation, with expected trends compared to total energy and launch pitch angle. With a table $\mathbb{T}(\mathsf{E}, \alpha)$, we proceed to reverse the velocities and launch distributions downward, towards our 'detector'.

The distribution is imposed at the top (lowest B) on total velocity, $|v|$ (i.e. we assume $T_{\parallel} = T_{\perp}$ and a flat distribution across pitch-angle space), and sampled at the test velocities used in $\mathbb{S}$. The environment here is assumed to be both homogeneous and large enough that any generated region of velocity-space at the top will detected at the bottom. Thus, we can neglect the x and y positions of particles in the simulation.

We seek to simulate an instrument observing the electrons being emitted effectively continuously from a source region, which contains a stable background distribution, and has a strong beam appear, stabilize, and then disappear. Thus, we set a Δt_D period over which the detector bins incoming particles, and a Δt_S period between source distribution 'launches'. To achieve something approaching the appearance of a continuous source, Δt_D should be at least $10\Delta t_S$.

A secondary background distribution serves as the main background to our beam for growth-rate calculation purposes. It is a localized population, distinct from the much colder, higher-density ionospheric background, and is composed of a population of degraded beam particles with a higher density and temperature than the beam population. We define it as a constant Maxwellian,

$$f(|v|) = n_e\,(2\pi)^{3/2}\;v_{th}^3\;\mathrm{e}^{-\left(\frac{v}{2v_{th}}\right)^2} = n_{bg}\left(\frac{m_e}{2\pi k T_{bg}}\right)^{3/2}\mathrm{e}^{-\frac{m_e(v)^2}{2kT_{bg}}} = f_{bg},$$

where, for a given input temperature, the electron number density n_{bg} is interpolated from a table of values used by *Lotko and Maggs* (1981).

The beam is built with a similar form, except for time-varying parameters $T_{beam}(t)$, $n_{beam}(t)$, and a velocity shift $\delta(t)$:

$$f_{beam}(|v|,t) = n_{beam}(t)\left(\frac{m_e}{2\pi k T_{beam}(t)}\right)^{3/2}\mathrm{e}^{-\frac{m_e(v-\delta(t))^2}{2kT_{beam}(t)}},$$

where in practice $T_{beam}(t)$ and $n_{beam}(t)$ are set as fractions of the secondary background values.

The final distribution is then the sum of these, as in Figure 3.18,

$$f(v,t) = f(v_\parallel, v_\perp, \theta, t) = f_{iono} + f_{bg} + f_{n_{beam}}(t)$$

where the contribution from the ionospheric background f_{iono} with a 2000 K temperature is included for completeness, but its effect on the distribution function is vanishingly small at these high energies.

To dimensionally reduce these towards a parallel distribution function $f(v_\parallel)$, we first sum over the azimuthal angles. This is not a simple sum: as these are finite cells in velocity-space, we must weight each angular 'wedge' by its accompanying $\Delta\theta_i$, i.e.

$$f(v_\parallel, v_\perp, t) = \sum_{i\forall\theta} f(v_\parallel, v_\perp, \theta_i, t)\Delta\theta_i,$$

where $\Delta\theta_i$ is set by the pitch angle, as in the hemispheric interpolation.

We put off dealing with time until now because the hemispheric interpolation introduces no time dependence, and so the azimuthal sum has none either. The next step will be interpolating and reducing away a dimension from our test-particle simulation, so we must

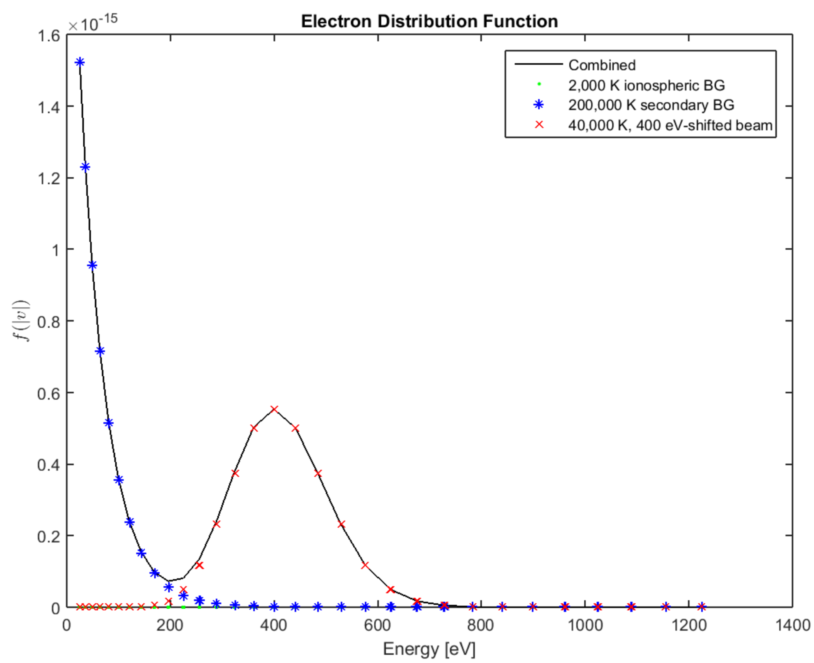

Figure 3.18: Imposed top distribution in $|v|$, showing ionospheric background distribution (green dots), secondary background (blue stars), a beam distribution (red Xs), and final combined sum (solid black line), with $T_{iono} = 2e3$ K, $n_e \sim 1.984e9$ m^{-3}, $T_{bg} = 2e5$ K, $n_{bg} = 1.066e6$ m^{-3}, $T_{beam} = 4e4$ K, $n_{beam} = 21.32e3$ m^{-3}, and $\delta = 400eV$.

take travel times into consideration beforehand. This is simply a set limitation at detector timeslice τ, such that the particles we consider are, henceforth, in the set $\mathbb{J}_\tau$ of particles whose launch time t_0 and travel time t_T fulfill $t_0 + t_T <= \tau$ and $> \tau - t_D$. There is also an implicit sum here as we are taking the detector integration into account, and this needs its own weighting value for f, simply the ratio of the launch time and the integration time, i.e.

$$f(v_\parallel, v_\perp, \tau) = \frac{\Delta t_S}{\Delta t_D} f(\mathbb{J}_\tau).$$

Next, we sum over perpendicular velocities to get a one-dimensional reduced distribution function, taken from the standard Landau theory for parallel propagation (e.g. *Ergun et al.* (1993)). This is slightly complicated by the large number of unique $v_\parallel$ values. We define a set of 'center points' in $v_\parallel$ as simply the points along the $v_\perp = 0$ axis. For each of these $v_{\parallel,i}$, all values of f with $v_\parallel$ in the range $\mu_\parallel = \left\{ \frac{v_{\parallel,i-1}+v_{\parallel,i}}{2}, \frac{v_{\parallel,i}+v_{\parallel,i+1}}{2} \right\}$ are summed over $v_\perp$ using a modification of the trapezoidal rule,

$$f(v_{\parallel,i}, \tau) = \sum_{j \forall v_\perp; \mu_\parallel} (v_{\perp,j+1} - v_{\perp,j}) \frac{f(\mu_\parallel, v_{\perp,j}, \tau) + f(\mu_\parallel, v_{\perp,j+1}, \tau)}{2} v_{\perp,j},$$

where the factor $v_{\perp,j}$ is the phase-space cell weighting. Figure 3.19 shows a color plot of the reduced distribution function vs. $v_\parallel$ and time, as well as several timeslices as the distribution evolves through the beam-arrival phase.

Now, with a time-integrated one-dimensional reduced distribution function in $v_\parallel$, we can calculate growth rates. For a given cold ionospheric background plasma frequency ω_p, wave vector $k = k_\parallel$, and test plasma frequency ω_t, the growth rate is

$$\gamma(f(v_\parallel), k, \omega_p, \omega_t, \tau) = \left(\frac{d\varepsilon}{d\omega} \right)^{-1} \mathrm{Sign}[k] \; \frac{\pi \omega_p^2}{k^2 n_e} \; \left[\frac{\partial f(v_\parallel, \tau)}{\partial v_\parallel} \right]_{k v_\parallel = \omega_t},$$

where ε is the dielectric function, approximated as $1 - \frac{\omega_p^2}{\omega_t^2}$ for cold plasma. The derivative $\partial f_1 / \partial v_\parallel$ is calculated at a test velocity related to the beam parameters, specifically the closest $v_\parallel$ value to $\delta + T_{beam}/k_B$.

Ideally we would want to set Δt_D to match the Correlator's timeslices, 1 ms, and to allow the simulation to 'settle' for a long enough time between source changes that even the slowest particles reach the detector, approximately 14 s per change. However, given our above guideline that $\Delta t_S \leq \Delta t_D / 10$, this would require storage of prohibitive numbers of time-overlapping distributions. From Figure 3.17 we know that the majority of particles will have arrived within 5 seconds, so we use that as our settling time, and only use realistic Δt_D in special cases. Finally, we relate k values and ω_t values via an approximation of the warm plasma dispersion relation, $\omega = \omega_p + \frac{3}{2} k^2 v_{th}^2 / \omega_p$, where v_{th} is the background ionospheric thermal velocity $\sqrt{3kT_{iono}/m_e}$.

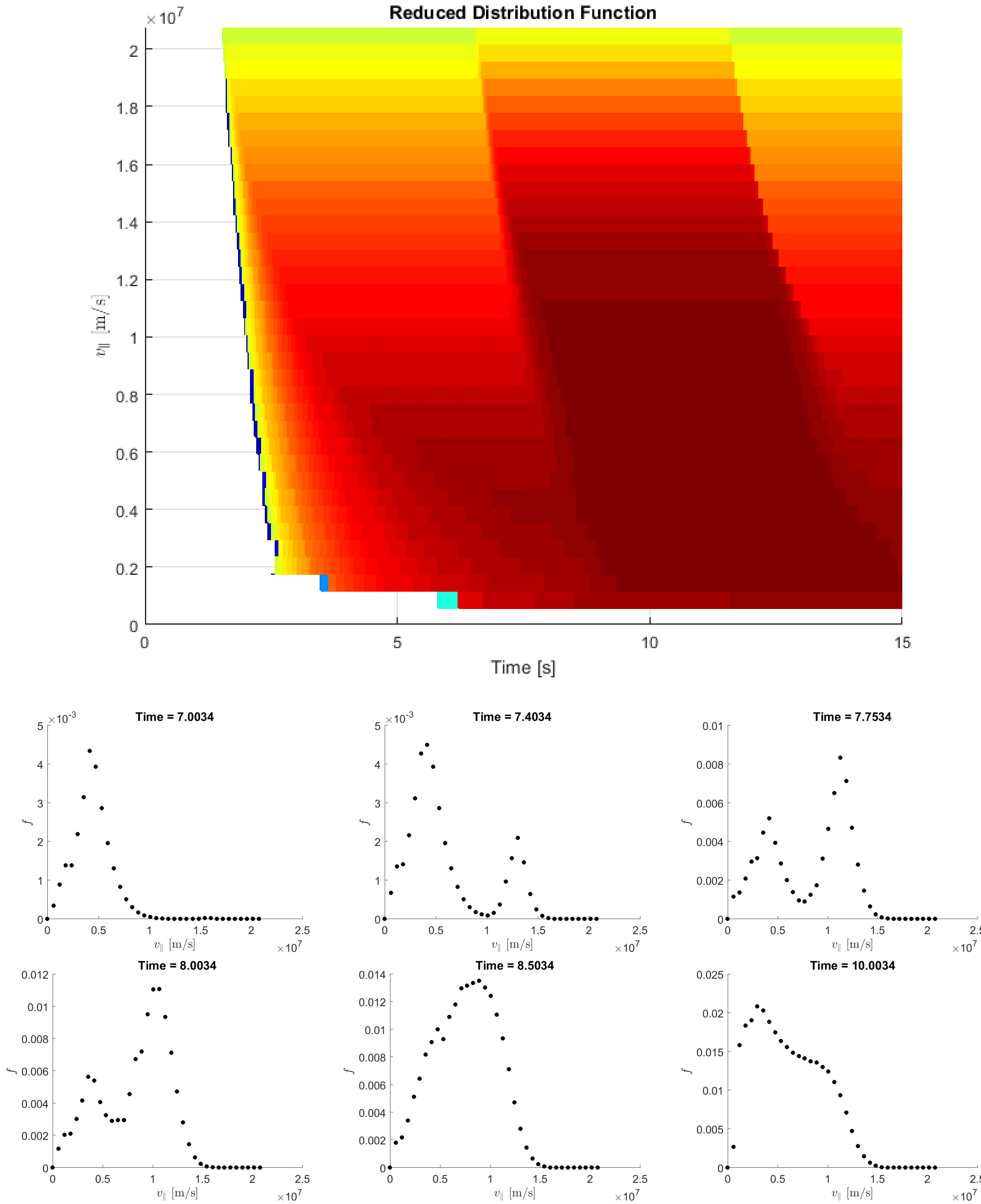

Figure 3.19: Reduced distribution function values. Top shows a color plot vs. $v_{\|}$ and time, for the entire time span of the test. Below, six timeslices from the beam arrival period, showing the formation and disappearance of a positive slope. Note that the continued appearance of low-energy particles near the bottom is the tail of the background distribution. It has not yet arrived before beam turn-on, but is considered too small to significantly affect the results.

Figure 3.20 shows the final results of the simulation: Langmuir wave growth rate γ, versus k and time (on the horizontal axis), calculated for ω_t from approximately $1.00022\omega_p$ to $1.0054\omega_p$. The top panel shows the n_{beam}/n_{bg} at the top, while the two columns are zoomed into the times at which the bulk of the particles arrive during beam turn-on (left), or depart with beam shutoff (right).

The top growth-rate panels show γ on the vertical axis, as well as in color scale (blue is negative, red positive). Both a growth rate spike during the beam arrival and a damping enhancement during beam departure are clearly visible. This result matches qualitatively the pattern observed in the phase of the in-phase, resistive component (I) of the wave-particle correlations during positive and negative gradients in beam flux.

In the lower panels, the color scale is still γ, and the vertical axis is the wavenumber k. The strongest growth and damping are associated with the long-wavelength modes at $k < 0.004$, which is generally expected as shorter-wavelength modes are more heavily damped. Growth at the long wavelengths is associated with the earliest-arriving, higher-energy particles, with the later lower-energy arrivals exciting some growth at shorter wavelengths.

The overall timeframe of the growth and damping peaks are of order 100 ms, which is significantly longer than most of the observed wave-particle correlation events; however, there is suggestion of shorter-scale time structure in the simulation events, particularly of a double-peak in the growth rate. To further probe the small-scale structure of these results, we can modify certain parameters of the distribution-building. Figure 3.21 shows the results of three such tests: moving the beam by half the inter-energy spacing, both up and down in energy, as well as removing half of the energies entirely. Motivated by the fact that realistic electron beams have lifetimes orders of magnitude smaller than the 5 s beams used in Figure 3.20, Figure 3.22 shows the result of a beam with identical parameters, but a lifetime of only 100 ms. The general impression of the above tests is that all of the small-scale structure seen in Figure 3.20 is heavily dependent on various aspects and limitations of the simulation system itself. Thus, these structures may bear little to no resemblance to any physical reality—even an approximation of such. Given this, as the simulation stands, no quantitative conclusions can follow regarding the small-timescale behavior of the growth rate.

3.6 Conclusions

The CHARM-II sounding rocket successfully carried a wave particle correlator to an apogee altitude of 802 km in substorm aurora. The correlator instrument locks onto the highest-amplitude waveform in the 100 kHz to 4 MHz range, ideally the wave at the Langmuir frequency, and bins incoming electrons at 8 energy levels into 16 phase bins. It returned data from > 400 seconds of flight time, and after both automated and manual event selection, 57 timeslices containing events of interest were selected and analyzed. Breakdown of the phase correlation data into resistive and reactive components revealed a striking relationship between electron beam dynamics and the nature of the wave-particle correlation: whenever the beam flux at the measured electron energy was increasing with time, the phase of the

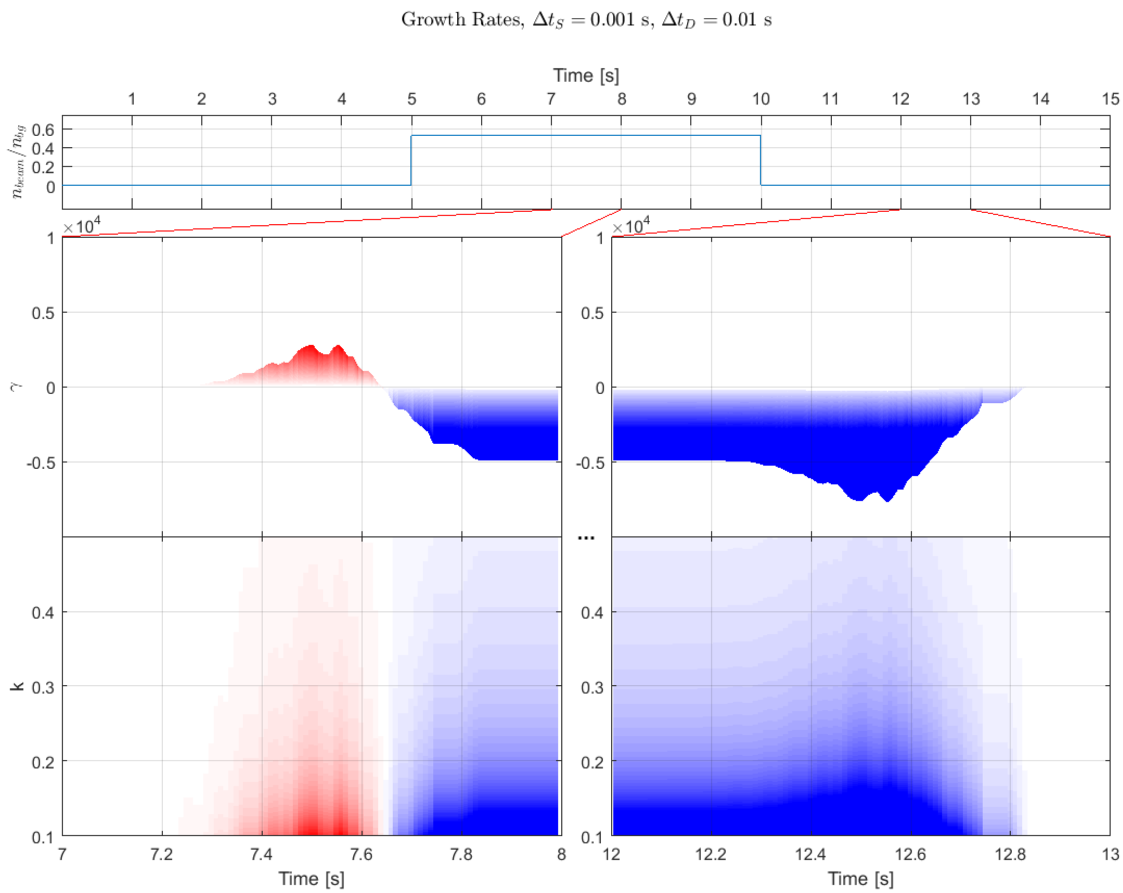

Figure 3.20: Results from the simulation: Langmuir wave growth rate γ, versus k and time (on the horizontal axis), calculated for ω_t from approximately $1.00022\omega_p$ to $1.0054\omega_p$. The top panel shows the n_{beam}/n_{bg} for at the top, while the two columns are zoomed into the times at which the bulk of the particles arrive during beam turn-on (left), or depart with beam shutoff (right). The top growth-rate panels show γ on the vertical axis, as well as in color scale (blue is negative, red positive), and both a growth rate spike during the beam arrival and a damping enhancement during beam departure are clearly visible, qualitatively matching the the pattern observed in the data. In the lower panels, the color scale is still γ, and the vertical axis is the wavenumber k.

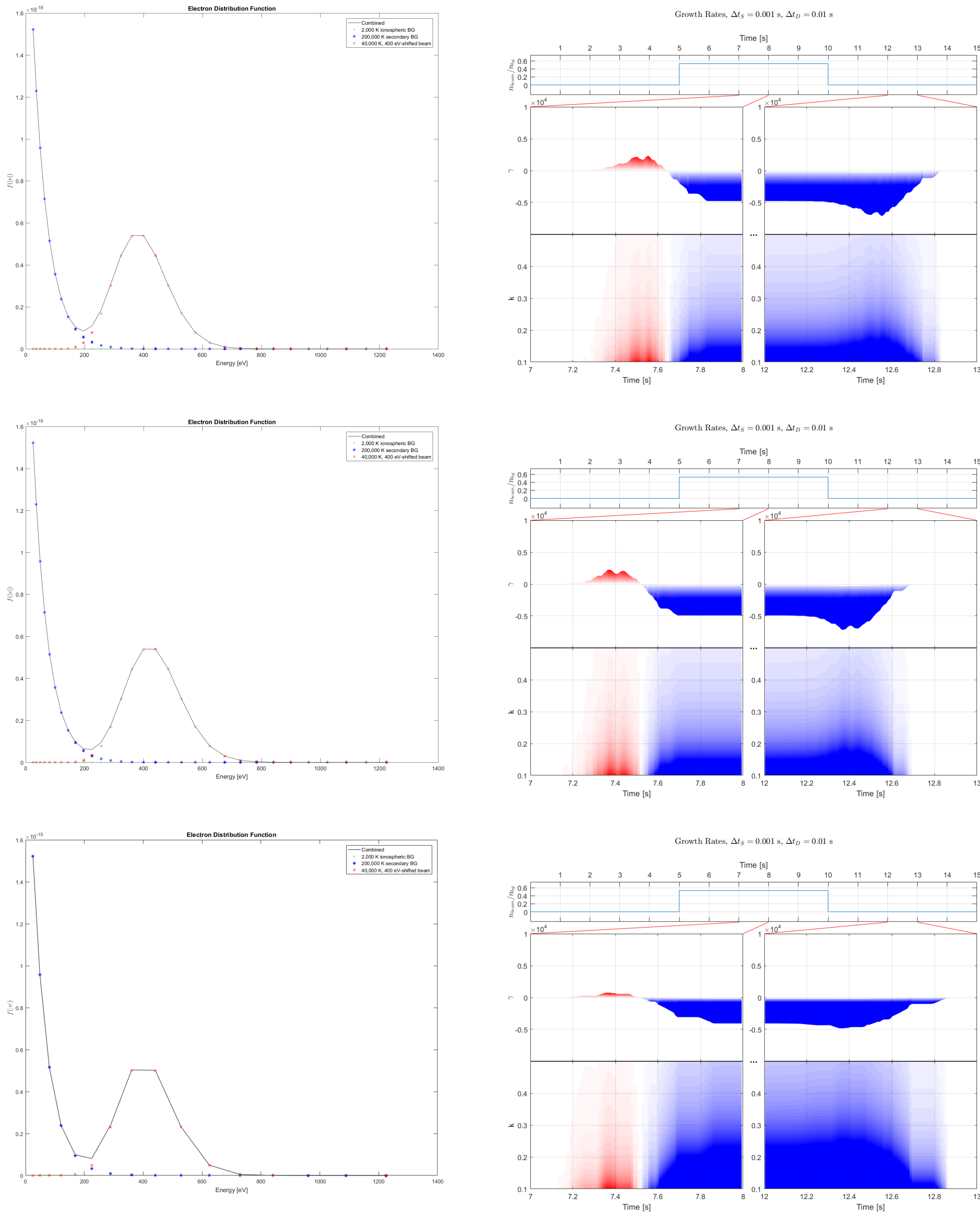

Figure 3.21: Test inputs and resultant growth rates, showing that much of the small-timescale structure seen in the growth rate is due to binning effects.

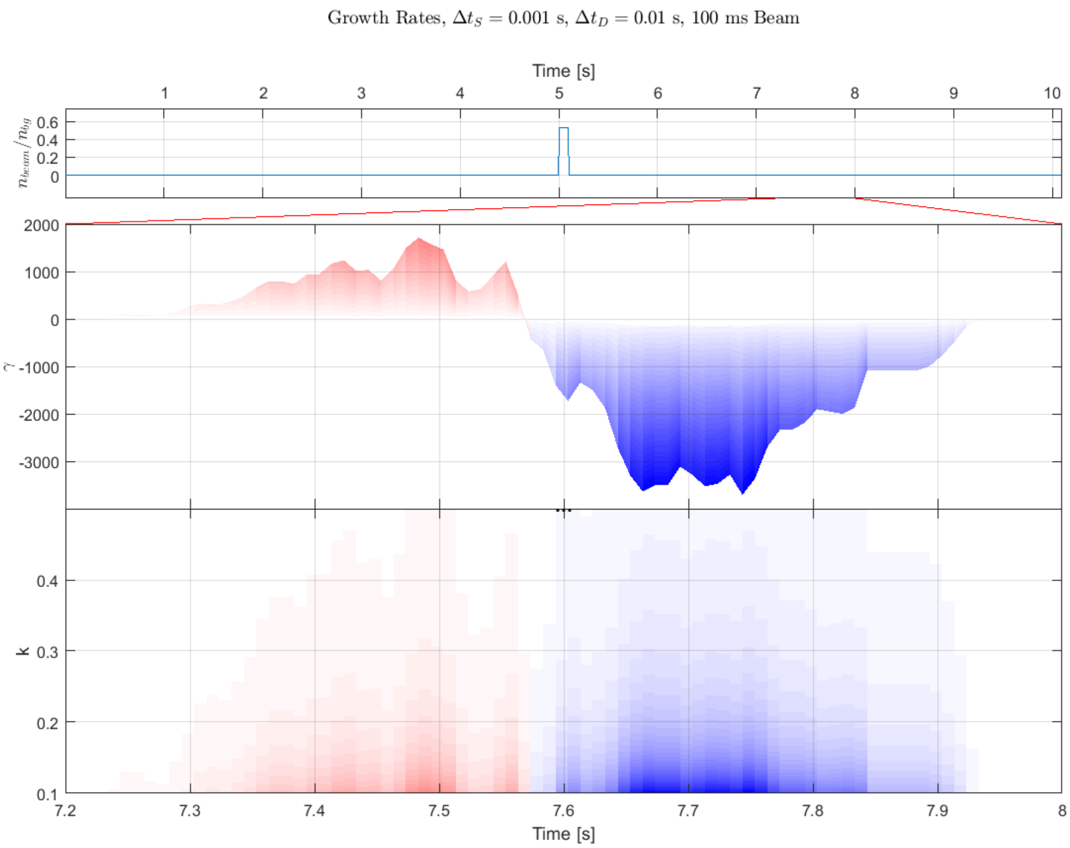

Figure 3.22: Growth rates resulting from a beam identical to that in Figure 3.20, but lasting only 1/50th of the time.

resistive component of the electron bunching implied energy transfer from the wave field to the particles, and when the electron beam flux was decreasing, the reverse occurred. This pattern was repeated for all events, and was particularly clear in several events, including the largest-amplitude event investigated by *Kletzing et al.* (2011).

Two related theories to explain this observation have been explored, one invoking the changing nature of the interactions of the electrons with a given Langmuir wave as the beam energy decreases, as typically occurs due to dispersion of an auroral electron beam accelerated several thousand kilometers above the interaction location; and the other invoking detailed features of the electron distribution function at ionospheric altitudes, arising when the electron beam is modulated at higher altitudes. A magneto-kinetic test-particle numerical simulation confirmed that for an electron beam which causes an impulsive increase in wave growth upon its appearance, its disappearance will be accompanied by an impulsive enhancement of wave damping within the same frequency range. The results therefore agree qualitatively with the experimental data from the CHARM-II rocket, though an exactly simulated quantitative representation has not yet been achieved.

The numerical simulation system developed consists of a flexible, cluster-enabled, node-independent suite of MATLAB scripts using the Boris Particle Pushing algorithm, modular background and beam distribution functions with arbitrary time-changing beam profiles, and adaptive distribution function dimensionality reducer algorithms. This system may be easily adapted to a multitude of tasks, including significant extensions of the work presented here, such as including non-uniform beam profiles and considering a broader class of Langmuir waves ($k_\perp \neq 0$) in the growth rate calculations. These and other extensions of the modeling may provide more quantitative tests of the time structure or magnitude of correlations observed in the CHARM-II wave particle correlator data.

Thanks to Dr. Scott Bounds and Dr. Stephen Kaeppler for assistance with the Correlator data and calibrations, and to Dr. Wayne Scales for providing the original magneto-kinetic test-particle code on which this simulation system was based.

Bibliography

Arnoldy, R. L., K. A. Lynch, and J. B. Austin, Energy and pitch angle-dispersed auroral electrons suggesting a time-variable, inverted-v potential structure, *J. Geophys. Res.*, *104*(A10), 22,613–22,621, 1999.

Birdsall, C. K., and A. B. Langdon, *Plasma Physics via Computer Simulation*, Institute of Physics Series in Plasma Physics, Taylor & Francis Group, New York, 2005.

Boehm, M., G. Paschmann, J. Clemmons, H. Höfner, R. Frenzel, M. Ertl, G. Haerendel, P. Hill, H. Lauche, L. Eliasson, and R. Lundin, The tesp electron spectrometer and correlator (f7) on freja, *Space Sci. Rev.*, *70*, 509–540, 1994.

Boehm, M. H., Waves and static electric fields in the auroral acceleration region, Ph.D. thesis, University of California, Berkeley, Berkeley, CA, 1987.

Boris, J. P., Relativistic plasma simulation-optimization of a hybrid code, in *Proceedings of the Conference on the Numerical Simulation of Plasmas (4th)*, edited by J. P. Boris and R. A. Shanny, 1, pp. 3–67, Plasma Physics Division, Naval Research Laboratory, National Technical Information Service, Washington, DC, 1970.

Cairns, I. H., The electron distribution function upstream of earth's bow shock, *J. Geophys. Res.*, *92*(A3), 2315–2327, 1987.

Chen, L. J., C. A. Kletzing, S. Hu, and S. R. Bounds, Auroral electron dispersion below inverted-v energies: Resonant deceleration and acceleration by alfvén waves, *J. Geophys. Res.*, *110*(A10), A10S13, doi:10.1029/2005JA011168, 2005.

Dombrowski, M. P., J. LaBelle, D. E. Rowland, R. F. Pfaff, and C. A. Kletzing, Interpretation of vector electric field measurements of bursty Langmuir waves in the cusp, *Journal of Geophysical Research (Space Physics)*, *117*, A09209, doi:10.1029/2012JA017741, 2012.

Ergun, R. E., C. W. Carlson, J. P. McFadden, and J. H. Clemmons, Langmuir wave growth and electron bunching: Results from a wave-particle correlator, *J. Geophys. Res.*, *96*(A1), 225–238, 1991a.

Ergun, R. E., C. W. Carlson, J. P. McFadden, D. M. TonThat, and J. H. Clemmons, Observation of electron bunching during landau growth and damping, *J. Geophys. Res.*, *96*(A7), 11,371–11,378, 1991b.

Ergun, R. E., G. T. Delory, E. Klementis, C. W. Carlson, J. P. McFadden, I. Roth, and

M. Temerin, Vlf wave growth from dispersive bursts of field-aligned electron fluxes, *J. Geophys. Res.*, *98*(A3), 3777–3787, 1993.

Ergun, R. E., J. P. McFadden, and C. W. Carlson, Wave-particle correlator instrument design, in *Measurement Techniques in Space Plasmas: Particles*, *Geophysical Monograph*, vol. Particles, pp. 325–331, American Geophysical Union, American Geophysical Union, 1998.

Ergun, R. E., D. M. Malaspina, I. H. Cairns, M. V. Goldman, D. L. Newman, P. A. Robinson, S. Eriksson, J. L. Bougeret, C. Briand, S. D. Bale, C. A. Cattell, P. J. Kellogg, and M. L. Kaiser, Eigeinmode structure in solar-wind langmuir waves, *Phys. Rev. Lett.*, *101*(051101), 2008.

Filbert, P. C., and P. J. Kellogg, Electrostatic noise at the plasma frequency, *J. Geophys. Res.*, *84*(A4), 1369–1381, 1979.

Gough, M. P., P. J. Christiansen, and K. Wilhelm, Auroral beam-plasma interactions: Particle correlator investigations, *J. Geophys. Res.*, *95*(A8), 12,287–12,294, 1990.

Gurnett, D. A., F. L. Scarf, W. S. Kurth, R. R. Shaw, and R. L. Poynter, Determination of jupiter's electron density profile from plasma wave observations, *J. Geophys. Res.*, *86*(A10), 8199–8212, 1981.

Kaeppler, S., C. A. Kletzing, S. R. Bounds, J. W. Gjerloev, B. J. Anderson, H. Korth, J. W. LaBelle, M. P. Dombrowski, M. Lessard, R. F. Pfaff, D. E. Rowland, S. Jones, and C. J. Heinselman, Current closure in the auroral ionosphere: Results from the auroral current and electrodynamics structure rocket mission, submitted Nov 2011, 2011.

Kintner, P. M., J. Bonnell, S. Powell, and J. E. Wahlund, First results from the freja hf snapshot receiver, *Geophys. Res. Lett.*, *22*, 287, 1995.

Kletzing, C. A., and S. Hu, Alfvén wave generated electron time dispersion, *Geophys. Res. Lett.*, *28*(4), 693–696, doi:10.1029/2000GL012179, 2001.

Kletzing, C. A., and L. Muschietti, *Phase Correlation of Electrons and Langmuir Waves*, *Lecture Notes in Physics*, vol. 687, chap. 13, pp. 313–337, Springer Berlin Heidelberg, 2006.

Kletzing, C. A., S. R. Bounds, J. LaBelle, and M. Samara, Observation of the reactive component of langmuir wave phase-bunched electrons, *Geophys. Res. Lett.*, *32*(L05106), doi:10.1029/2004GL021175, 2005.

Kletzing, C. A., S. R. Kaeppler, S. R. Bounds, J. LaBelle, and M. P. Dombrowski, Nonlinear wave evolution: Observation of electron phase bunching in auroral langmuir waves, Abstract NG23A-1478 presented at 2011 Fall Meeting, AGU, San Francisco, Calif., 5-9 Dec., 2011.

Langmuir, I., Oscillations in ionized gasses, *Proceedings of the National Academy of Sciences*, *14*(8), 627–637, 1928.

Lin, R. P., D. W. Potter, D. A. Gurnett, and F. L. Scarf, Energetic electrons and plasma waves associated with a solar type iii radio burst, *Astrophys. J.*, *251*(1), 364, 1981.

Lotko, W., and J. E. Maggs, Amplification of electrostatic noise in cyclotron resonance with an adiabatic auroral beam, *J. Geophys. Res.*, *86*(A5), 3449–3458, 1981.

Malaspina, D. M., and R. E. Ergun, Observations of three-dimensional langmuir wave structure, *J. Geophys. Res.*, *113*(A12108), 2008.

Malaspina, D. M., I. H. Cairns, and R. E. Ergun, Antenna radiation near the local plasma frequency by langmuir wave eigenmodes, *Astrophys. J.*, *755*(45), 2012.

McAdams, K. L., Sounding rocket based investigations of hf waves in the auroral ionosphere, Ph.D. thesis, Dartmouth College, 6127 Wilder Laboratory, Hanover, NH 03755, 1999.

McAdams, K. L., J. LaBelle, P. W. Schuck, and P. M. Kintner, Phaze ii observations of lower hybrid burst structures occurring on density gradients, *Geophys. Res. Lett.*, *25*(16), 3091–3094, 1998.

McAdams, K. L., R. E. Ergun, and J. LaBelle, Hf chirps: Eigenmode trapping in density depletions, *Geophys. Res. Lett.*, *27*(3), 321–324, 2000.

Muschietti, L., I. Roth, and R. E. Ergun, Interaction of langmuir wave packets with streaming electrons: Phase-correlation aspects, *Phys. Plasmas*, *1*, 1008, 1994.

Samara, M., Sounding rocket based investigations of whister, upper hybrid and langmuir waves in the auroral ionosphere, Ph.D. thesis, Dartmouth College, 6127 Wilder Laboratory, Hanover, NH 03755, 2005.

Samara, M., and J. LaBelle, Lf/mf whistler mode dispersive signals observed with rocket-borne instruments in the auroral downward current region, *J. Geophys. Res.*, *111*(A09305), 2006.

Samara, M., J. LaBelle, C. A. Kletzing, and S. R. Bounds, Rocket observations of structured upper hybrid waves at fuh = 2fce, *Geophys. Res. Lett.*, *31*(22), 2004.

Chapter 4

Bursty Langmuir Waves in the Cusp: TRICE

4.1 Introduction

The electron-beam plasma interaction in the Earth's auroral ionosphere produces a variety of plasma waves, and Langmuir waves are among the most intense and ubiquitous of these. Sounding rocket observations of Langmuir waves in both the night- and dayside aurora show similar features: the waves occur in bursts with durations from ms to hundreds of ms and amplitudes from mV/m to hundreds of mV/m (*Boehm* 1987; *McFadden et al.* 1986; *McAdams et al.* 1999). The bursts are modulated at frequencies ranging from < 1 kHz to > 50 kHz (*Ergun et al.* 1991; *Bonnell et al.* 1997; *LaBelle et al.* 2010). Satellite data confirm many of these observations (*Gurnett et al.* 1981; *Beghin et al.* 1981; *Stasiewicz et al.* 1996; *Kintner et al.* 1996; *Malaspina and Ergun* 2008).

The modulation of auroral Langmuir waves has been attributed to mixing of multiple wave normal modes on the Langmuir surface. Indeed, spectra of the modulated waves show multiple peaks, extremely well resolved in recent experiments (*LaBelle et al.* 2010). Linear (*Maggs* 1976; *Newman et al.* 1994b) and nonlinear (*Newman et al.* 1994a) theory suggests that the auroral electron beam can excite a range of Langmuir modes, including modes at oblique angles. There is some controversy, however, about the origin of the interfering waves. Most papers in the literature favor wave-wave interaction, whereby a primary Langmuir wave, directly excited by the beam, decays into a second Langmuir wave and a whistler or ion sound wave, with the observed modulation resulting from the beating of the two Langmuir waves (*Bonnell et al.* 1997; *Stasiewicz et al.* 1996; *Lizunov et al.* 2001; *Khotyaintsev et al.* 2001). The absence of evidence for the low frequency wave in many observations is explained by strong wave damping. An alternative hypothesis holds that two different Langmuir waves directly excited by linear processes at slightly different locations in the strongly inhomogeneous plasma mix to make the observed modulations (*LaBelle et al.* 2010). In this case, no third wave would be expected to occur.

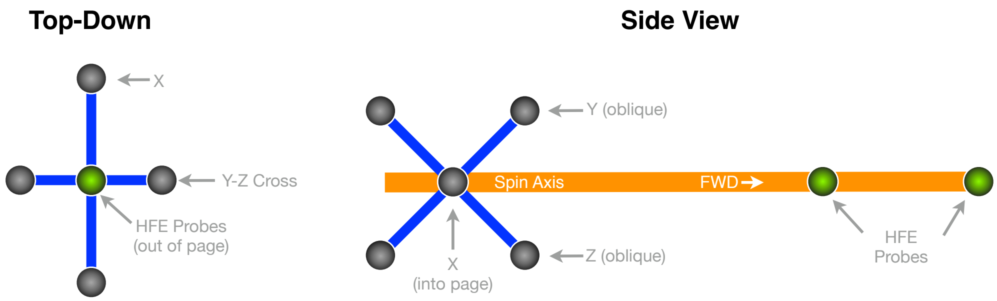

Figure 4.1: Boom and sensor elements of the two complimentary HF electric field experiments flown on the TRICE sounding rockets.

A significant shortcoming of previous rocket experiments lies in the absence of three-dimensional field data at high frequencies. Recently, however, the availability of higher rocket telemetry rates and the increasing power of onboard processing enabled the TRICE sounding rocket mission to include such measurements. In principle, having all three components of the electric field provides more information about the wave modes comprising the modulated Langmuir waves. Section 4.2 below describes the TRICE instrumentation; Section 4.3 shows measured three-dimensional waveforms; and Section 4.4 presents a model for explaining observed effects. Finally, Section 4.5 expands upon an ambiguity in the model.

4.2 Instrumentation

The Twin Rockets to Investigate Cusp Electrodynamics (TRICE) were launched 10 Dec 2007 at 0900 and 0902 UT, from Andoya, Norway, reaching apogees of 1145 km and 750 km. The rockets were launched into an active cusp, with poleward-moving auroral forms monitored with all-sky cameras and multiple radars. K_p was relatively low, with a value of two. The payload's spin axis was kept aligned with the background magnetic field, B_0, by a NASA Attitude Control System (ACS), which activated at specific times during the flight, and was otherwise turned off to minimize interference. Each rocket carried electron and ion detectors, swept Langmuir probes, low-frequency to DC electric-field probes, magnetometers, and two complementary high-frequency electric-field instruments: the Dartmouth High-Frequency Electric-field experiment (HFE), and the NASA GSFC Tri-Axial Electric-Field Wave Detector (TAEFWD). Figure 4.1 shows the probe configuration for the HF instruments.

The Dartmouth HFE detects the potential difference between two 2.5 cm spherical probes, separated by 30 cm along the payload's spin axis. This ΔV signal provides an estimate of the the axial component of the electric-field, which is mainly parallel to the ambient magnetic field, given the payload alignment. The signal is band-pass filtered to the 100 kHz to 5 MHz band, and regulated by an Automatic Gain Control (AGC) system to enhance the dynamic range. The AGC control signal is sampled onboard at 20 kHz and telemetered with other digital data. The regulated HF signal directly modulates a 5 MHz-bandwidth S-band transmitter, and the resulting waveform is continuously digitized at the ground telemetry station at 10 MHz, with 12-bit resolution. This instrument is the latest iteration of a design which has flown on numerous rocket campaigns in both $\boldsymbol{E}_{\parallel}$ and $\boldsymbol{E}_{\perp}$ configurations, including HIBAR (*Samara et al.* 2004), PHAZE II (*McAdams et al.* 1998), SIERRA, RACE (*Samara and LaBelle* 2006), ACES (*Kaeppler et al.* 2011), and CHARM II (*Kletzing et al.* 2012).

The GSFC TAEFWD measures ΔV between three pairs of 2.5 cm probes separated by 47.5 cm (X axis) or 45.5 cm (Y and Z axes) along three orthogonal axes: an X axis perpendicular to the payload spin axis, and Y and Z axes 45 degrees off of the spin axis, in a plane perpendicular to X. After filtering to a 4 MHz bandwidth (-3dB bandwidth), the onboard TAEFWD receiver synchronously digitizes these three signals, as well as the Dartmouth HFE output signal, at 8 MSps, in 2048-sample snapshots, with the snapshots being gathered at 15.625 Hz cadence, yielding a 0.4% duty cycle and a 1.92 MBps data stream (after packing into 10-bit words). The sensitivity level was approximately 80 μV/m, suitable for measurement of large-amplitude Langmuir waves in the cusp region.

Both payloads were affected by various payload systems failures and instrumental anomalies, resulting in the complete loss of particle data and interference in other data, including the generation of extraneous signals; however, the DC magnetometer and both HF electric-field experiments obtained good data over most of the flight. The HFE data in particular resulted in a study of cusp Langmuir waves (*LaBelle et al.* 2010).

Between ACS activations, imperfections in the payload weight distribution caused the payload to begin to misalign from $\boldsymbol{B}_0$, with the spin axis precessing around $\boldsymbol{B}_0$ by an increasing angle—an effect known as 'coning'. On TRICE, unlike most flights, this effect could not be compensated for because the payload attitude data provided by the ACS package was of low quality due to interference. Over the interval important to this study, the payload magnetometer showed a variance in $\boldsymbol{B}_0$ from perfect spin-axis alignment of 5-10%, implying a similar variance in the components of the electric field parallel and perpendicular to $\boldsymbol{B}_0$.

4.3 Observations

As shown by *LaBelle et al.* (2010), the TRICE high-flyer passed through auroral activity that generated significant high-frequency waves. Figure 4.2a is a 0 to 2.5 MHz spectrogram of HFE data from almost the entire flight (100 to 1100 seconds after launch). General features of this plot include: a 100 kHz rolloff due to the band-pass filter; vertical bands which are caused by the AGC system raising and lowering the noise floor when the total signal amplitude changes; additional, cadenced vertical bands at 60 s intervals, which are instrument-calibration signals injected into the receiver; and multiple horizontal bands, the strongest of which are due to interference from payload systems. Malfunctions in the particle instruments also resulted in periods of strong interference 100–400 kHz, e.g. at 710–845 s, 865–880 s, and 900–920 s. These broadly resemble natural auroral hiss, but have been attributed to arcing in high-voltage components (*LaBelle et al.* 2010).

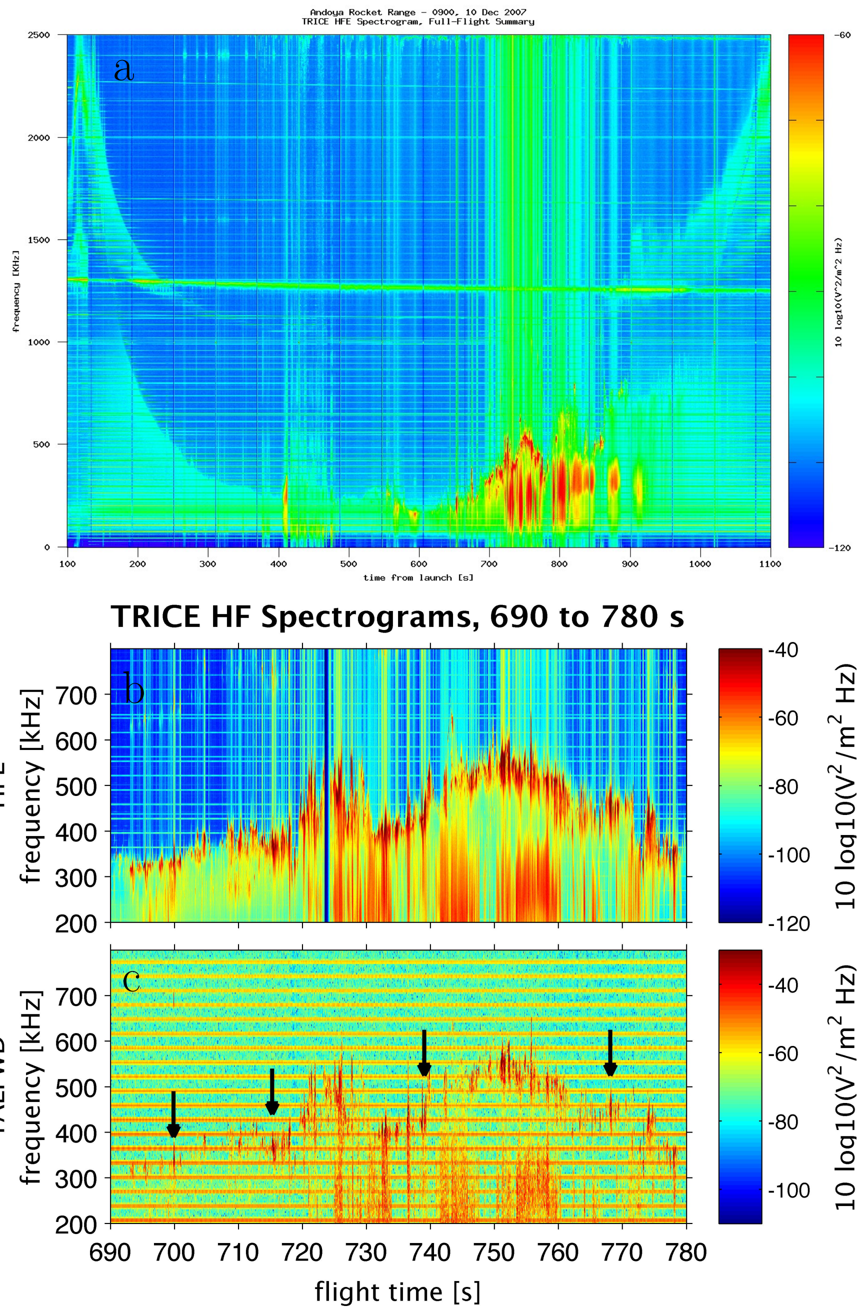

Figure 4.2: Spectrograms of TRICE HF electric field data. **a** shows a summary of the full flight covering 100 kHz–2.5 MHz and 100–1100 s, while **b** and **c** are HFE and TAEFWD Z' data covering a 200-800 kHz band from 690 to 780 s, the period of intense activity outlined in white in **a**. Black arrows in **c** indicate times of TAEFWD waveform snapshots examined in detail.

Besides the artificial features, the full-flight spectrogram shows signals of natural origin, such as wave cutoffs of the type which have proven effective for measuring electron plasma density on previous flights. Two such upper cutoffs can be seen near the beginning of this flight: a higher-frequency cutoff that emerges from diffuse noise near 2.5 MHz at 150 s and descends to around 1.2 MHz by 300 s, and a lower-frequency cutoff which is at 750 kHz at 200 s, 200 kHz at 400 s, and remains reasonably well-defined through most of the flight. This lower-frequency cutoff is interpreted as f_{pe}, which is an upper bound for the whistler mode during the portion of the flight when $f_{pe} < f_{ce}$, approximately 190 to 900 s. The upper cutoff is identified as the upper-hybrid frequency $f_{uh} = \sqrt{f_{pe}^2 + f_{ce}^2}$, given that f_{ce} is approximately 1.2 MHz throughout the flight. The relation between the frequencies of these cutoffs when they occur together lends confidence to their interpretations as f_{pe} and f_{ce}.

The second half of the flight includes lengthy periods of intense wave activity near the f_{pe} cutoff, and close inspection of waveforms shows that these consist of many bursts of Langmuir waves. *LaBelle et al.* (2010) investigated 41 bursts occurring during the 850–861 s interval. They estimated that over 1000 bursts occurred over the entire flight, with durations from 20 to 250 ms, amplitudes ranging from a few mV/m to nearly 1 V/m, and with modulation frequencies ranging from less than 1 kHz to over 50 kHz.

The period from 690 to 780 s, outlined in white in Figure 4.2a, showed strong activity on both HF electric-field instruments. This period is expanded in Figure 4.2b for the HFE, and Figure 4.2c from the TAEFWD Z' channel. This channel (and its counterpart Y') is a composite channel derived by taking a linear combination of the real TAEFWD Y and Z channels, such that Z' is parallel to the spin axis (and thus the HFE boom), and Y' is orthogonal to Z' and X. The TAEFWD is somewhat less sensitive than the HFE, and suffers from some instrumental interference which produces the horizontal bands in Figure 4.2c. Nevertheless, after eliminating weak bursts which were dominated by interference or showed no clear modulation, approximately 50 clean TAEFWD snapshots with examples of bursty Langmuir waves were identified in this time period.

Figure 4.3 shows four selected 3-channel, 2048-sample, 0.256 ms TAEFWD snapshots corresponding to the times indicated by arrows in Figure 4.2c. In each example, the top, middle, and bottom panels show the X, Y', and Z' components of the HF electric field, respectively. In all of the waveforms in Figure 4.3, interference appears as discontinuous pulses at approximately 25 μs intervals; however, for the selected snapshots, the wave amplitude is high enough that these interference spikes do not affect identification of peaks and nulls in the wave modulation. All channels were normalized such that the Z' component derived from TAEFWD data was equal in magnitude to the axial component of the HF electric field derived from HFE data, which was in turn converted to absolute electric-field units using pre-launch HFE calibrations, with an adjustment factor to account for probe–plasma sheath capacitance.

Figure 4.3a shows waveforms from within a typical burst of cusp Langmuir waves, occurring at 699.8523 s. The variations in amplitude represent the Langmuir wave modulation which has been studied by many authors (*Bonnell et al.* 1997; *Stasiewicz et al.* 1996; *LaBelle et al.* 2010). The typical modulation frequency is 10 kHz, or 0.1 ms, so the 0.256 ms duration

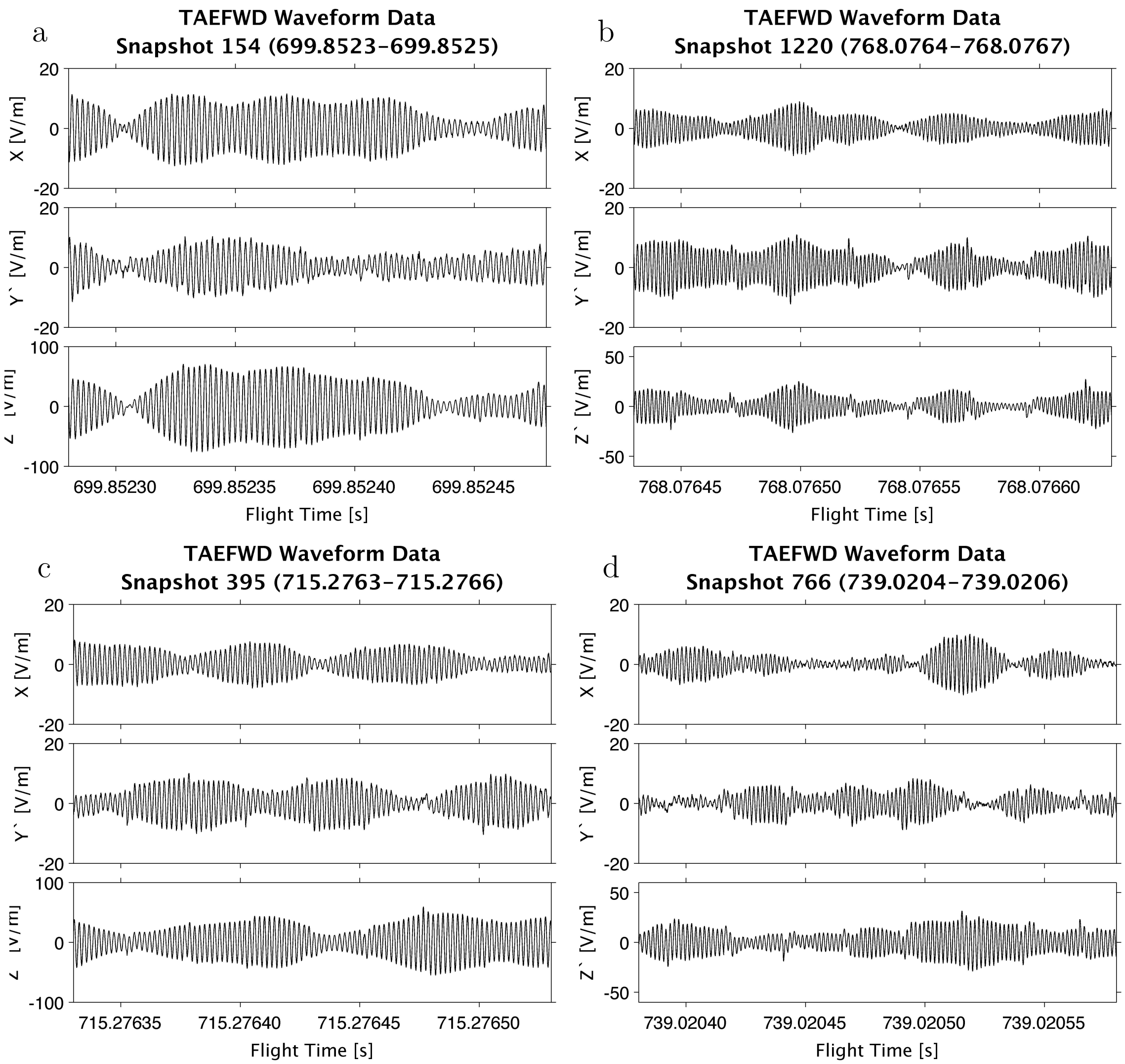

Figure 4.3: Waveform snapshots measured by the TAEFWD, for the four example Langmuir wave bursts indicated by arrows in Figure 4.2c. In **a** and **b**, the modulations of the mutually orthogonal components of wave electric field are synchronous. In **c** and **d** the modulation are not synchronous, indicating anisotropy.

TAEFWD snapshots catch only a small portion of the total modulated wave burst, implying that these plots show only one or a few modulation cycles out of many that occurred. In this example, the modulation nulls about 30 μs from the start of the snapshot are synchronous across all channels, i.e. the modulation of the x, y, and z components of the electric field are in-phase, and the burst modulation is relatively low-frequency ($\sim$ 15 kHz) and approximately monochromatic, implying sinusoidal modulation. Figure 4.3b from 768.0764 s shows an example with faster, multi-frequency modulation. This results in multiple nulls and peaks, which are variably spaced in time. As in the first example, the modulation is isotropic, i.e. in-phase on the three components of the wave electric field.

Of the snapshots with clear, high-powered bursts, up to 25% show modulation that appears to be anisotropic, i.e., the modulation nulls and peaks are not synchronous in the three electric-field components. Figure 4.3c from 715.2763 s shows an example of this behavior, with the modulation nulls and peaks coming at different times in different channels: e.g. a null appears in the X component at 715.276375 s, then in the Y' component 20 μs later, and finally in the Z' component after another 20 μs. In this case, the modulation appears similar in the three electric field components, but peaks and nulls are delayed. In other cases, the modulation of the wave electric field components is not only asynchronous, but the modulation appears entirely uncorrelated. Figure 4.3d shows an example of this behavior, from 739.0204 s. Starting from 739.02000 s, the nulls in this snapshot appear at 45 μs, 49.5 μs, 48.5 μs, and 58 μs in the X component, while the only clear nulls in the Y' component are at 39.5 μs and 52.5 μs, and the Z' component shows no clear nulls.

4.4 Wave Beating and Polarization

These first three-dimensional Langmuir-wave observations from a rocket show that, up to 25% of the time, the modulation of bursty Langmuir waves is anisotropic, meaning that the nulls and peaks are out of phase in the three electric field components. The superposition of two or more linearly polarized waves cannot produce such an effect. We postulate that the anisotropic modulations observed in TRICE high-flyer TAEFWD data result from mixing of Langmuir/whistler-mode waves with a variety of polarizations. To test the plausibility of this postulate, we model the superposition of combinations of wave normal modes which occur in a plasma similar to that encountered by the TRICE high-flyer from 690 to 780 s.

Figure 4.4 shows the result of numerical calculations of wave dispersion characteristics for high-frequency waves in the ionosphere. This dispersion surface was calculated using J-WHAMP, a Java-enhanced version of the Waves in Homogeneous Anisotropic Multicomponent Plasmas (WHAMP) program developed by *Rönnmark* (1982). This code uses numerical approximations of linear Vlasov theory to map out dispersion relations for a given set of plasma and environmental parameters, returning the basic characteristics of the normal modes, such as frequency (ω), wavenumber parallel to the ambient magnetic field ($k_{\|}$), and wavenumber perpendicular to B ($k_{\perp}$). It also returns many additional plasma, wave, and field characteristics, such as Alfvén speed, polarization, Stokes parameters, etc.

The plotted surface in Figure 4.4 is the locus of frequencies and wave vectors corresponding

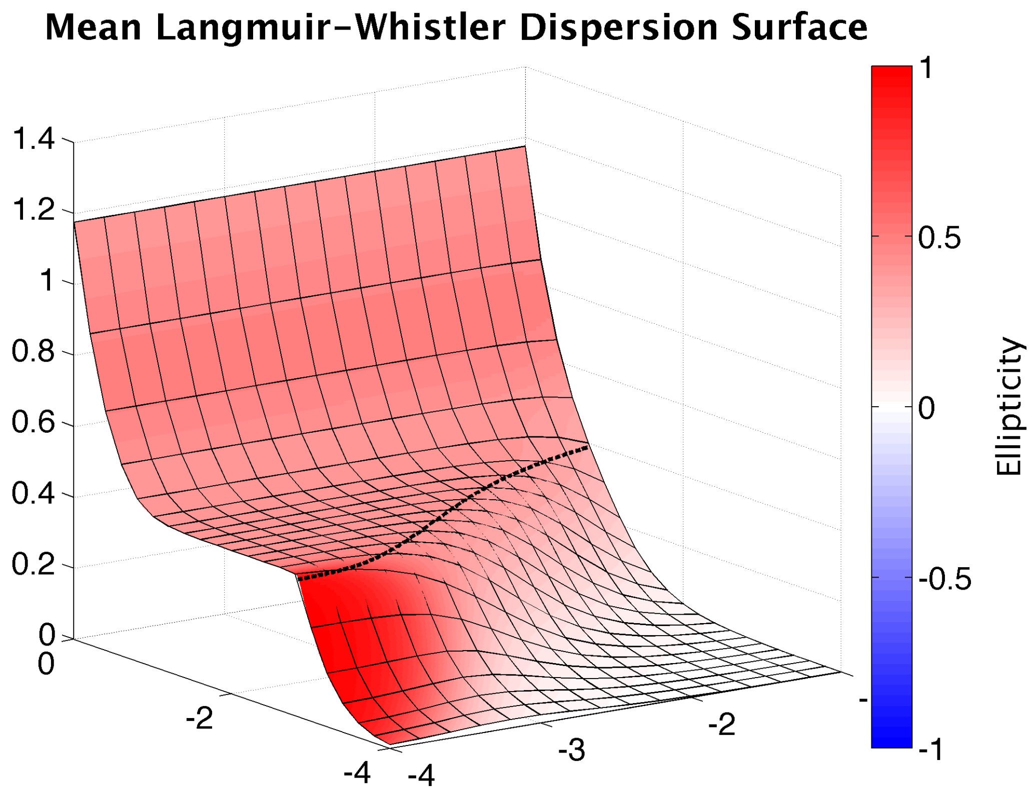

Figure 4.4: A dispersion relation for Langmuir and whistler waves in a homogeneous, magnetized plasma, calculated using the J-WHAMP numerical code. The surface shows frequency normalized to ω_{ce} vs. parallel and perpendicular wavevector (log scales). The model plasma approximately matches the conditions encountered by the TRICE high-flyer from 690 to 780 s. Shading of the surface represents the ellipticity of the modes, which range from left elliptically polarized (blue, not present on this surface) through linear (white) to right elliptically polarized (red). The dashed line is a example contour of constant frequency.

to the normal modes of the plasma. The x and y axes represent the logarithm of $k_\perp$ and $k_\parallel$, respectively, with each normalized to the electron gyroradius ρ_e. The vertical axis shows ω at a given ($k_\perp$, $k_\parallel$), normalized to the electron cyclotron frequency. The dashed line shows a constant-frequency contour, just below f_{pe}. The parameters used to generate this surface were selected to match plasma conditions encountered by the TRICE high-flyer during the 690 to 780 s period: $B_0 = 36.850$ μT (implying $f_{ce} = 1031.800$ kHz), and a single particle species set for electron parameters, with $n = 2149$ cm^{-3} (implying $f_{pe} = 417$ kHz) and an isotropic temperature of 2 eV with no loss cone. The model plasma includes only thermal (background ionosphere) electrons, because this population is what determines the real part of the wave dispersion relation, i.e. the frequencies, wave vectors, and polarization of the normal modes. At these frequencies, ions are a static background, with no significant effect. A more-complex electron beam model would be required to calculate the imaginary part of the dispersion relation (e.g. to examine damping rates), but is not required for our investigation of interference as a function of wave polarization, as an unrealistically high-density beam would be required to affect the mode structure and polarization. In the aurora, beam densities are typically 10^{-3} smaller than the ambient electron density. The significant J-WHAMP outputs for this analysis are those related to the polarization of normal modes: electric-field coefficients for generated waves, and the ellipticity parameter, which is a measure of polarization ranging from -1 (left-circularly polarized) through 0 (linearly polarized) to +1 (right-circularly polarized). In Figure 4.4, the ellipticity is represented by the color of the dispersion surface.

The dispersion characteristics of Figure 4.4 are similar to those calculated by *André* (1985), specifically his 'model 2' for an $f_{pe} < f_{ce}$ regime. In the limit $k_\parallel \gtrsim k_\perp$ and $k_\parallel \rho_e >> 0.001$, the surface corresponds to Langmuir waves, for which $\omega \approx \omega_{pe}$ at intermediate k. Dispersion sets in at short wavelengths ($k_\parallel \rho_e > 0.1$) due to finite electron temperature effects. For $k_\parallel >> k_\perp$ and long wavelengths in the $f_{pe} < f_{ce}$ regime, the Langmuir mode smoothly couples to the whistler mode, and the surface is better described as 'Langmuir-whistler modes' (*Layden et al.* 2011). These mode identifications can also be confirmed by the ellipticity: a region of strong right-elliptical polarization (REP) coincides with the whistler modes, while the Langmuir wave modes are more linearly polarized (LP).

The observations show modulated Langmuir waves which have been interpreted as wave beating due to the presence of multiple normal modes with closely spaced frequencies, e.g. waves near the 500 kHz plasma frequency with frequency separations of approximately 10 kHz. To investigate the intermodulation of such closely spaced normal modes, we can select normal modes from the Figure 4.4 data: one on a specific frequency contour and one near that contour (though not necessarily near the first point), representing waves with relatively similar frequencies. As shown by the dashed line in Figure 4.4, two modes thus selected can have very close frequencies, but significantly different wave vectors and polarizations. For example, one wave can be partway down the whistler dispersion curve, in the range of REP wave modes, and the other can lie in the mostly LP Langmuir wave region.

In order to simulate the superposition of two waves, we start by extracting five parameters—$k_\perp$, $k_\parallel$, and the complex E_x, E_y, and E_z coefficients—from the J-WHAMP output, for two selected normal modes from our dispersion surface. The E coefficients output by J-WHAMP

Table 4.1

Wave normal mode parameters[1].

#	ψ (deg)	LP /REP	f_{pe}/f_{ce}	E (x, y, z)	k $(x/\perp,\ y,\ z/\|)$	Ellip
a	45	LP	0.369469	(0.376424, 0.002857i, 0.926443)	(0.003162, 0, 0.007943)	0.3759
		LP	0.366606	(0.391707, 0.003042i, 0.920085)	(0.003236, 0, 0.007762)	0.3732
b	45	REP	0.368377	(0.630643, 0.512265i, 0.582988)	(0.000158, 0, 0.001380)	0.9093
		REP	0.369478	(0.632961, 0.524797i, 0.569165)	(0.000148, 0, 0.001380)	0.9176
c	45	LP	0.369469	(0.376424, 0.002857i, 0.926443)	(0.003162, 0, 0.007943)	0.3759
		REP	0.368377	(0.630642, 0.512265i, 0.582988)	(0.000158, 0, 0.001380)	0.9093
d	45	LP	0.366606	(0.391707, 0.003042i, 0.920085)	(0.003236, 0, 0.007762)	0.3732
		REP	0.369478	(0.632961, 0.524797i, 0.569165)	(0.000148, 0, 0.001380)	0.9176

[1] Determined from J-WHAMP, and used for the waveform simulations shown in Figure 4.5, along with the ellipticity for each mode. Model c uses the first waves from model a and model b, and models c and d differ only in the coordinate system rotation parameter ψ.

are normalized such that $|E| = 1$ mV/m, thus assuring that the interacting waves have comparable amplitudes, and modulation peaks and nulls should be at their most-pronounced. The parameters are plugged into the general plane wave equation summed over both waves

$$\vec{E}(t) = \begin{cases} \left(E_{x1}e^{-\mathbf{i}(k_1\cdot\vec{x}-\omega t)} + E_{x2}e^{-\mathbf{i}(k_2\cdot\vec{x}-\omega t)}\right)\hat{x}+ \\ \left(E_{y1}e^{-\mathbf{i}(k_1\cdot\vec{x}-\omega t)} + E_{y2}e^{-\mathbf{i}(k_2\cdot\vec{x}-\omega t)}\right)\hat{y}+ \\ \left(E_{z1}e^{-\mathbf{i}(k_1\cdot\vec{x}-\omega t)} + E_{z2}e^{-\mathbf{i}(k_2\cdot\vec{x}-\omega t)}\right)\hat{z} \end{cases} .$$

There exists an ambiguity in any dispersion solution which results in a rotational freedom around the $k_\parallel$ axis. J-WHAMP defines that $k_\parallel \equiv k_z$ and $k_\perp \equiv k_x$, implying that k_y must be zero—this is not a fully general solution. Section 4.5 examines this $k_\perp$ ambiguity with respect to J-WHAMP output, and finds that rotations around the z axis can generate some beat modulation anisotropy, but cannot fully explain the observations. Furthermore, in order to optimally pick up beating between the x and y components of waves, one must rotate all beating waves by an angle $\phi = 45$ degrees around the z axis. Figure 4.5 shows the results of the simulations, for four pairs of normal modes from the Langmuir plane, with $\phi = 45$ degrees. Using J-WHAMP definitions of the coordinates, the x, y, and z directions roughly correspond to those in the TAEFWD data, with z parallel to B_0, and x and y perpendicular. Table 4.1 lists the full parameters (f_{pe}, E, k, and ψ) used for the Figure 4.5 simulations.

Figure 4.5a shows the result of combining two simulated wave modes from the LP Langmuir dispersion region, at sufficiently short wavelengths for $\omega \sim \omega_{pe}$. As expected, the resulting modulation occurs with a 330 μs period ($1/\Delta f => 1/(3$ kHz)). In this case, corresponding to beating of two LP waves, the wave modulation is isotropic, i.e. maxima and minima coincide in time, similar to the observations shown in Figure 4.3a and b. Figure 4.5b shows the similar wave modulation that results from superposing two REP waves selected from the region $k_\parallel \sim 0.001$ and $k_\perp < 0.001$, which is within the whistler-mode part of the dispersion surface. The resulting modulation has a 660 μs period, and shows isotropic modulation.

In Figure 4.5c and d, the wave modes pairs from a and b were combined to make two mixed pairs in order to investigate beating between waves of different polarizations. This combination of one wave mode from each dispersion region yields modulation anisotropy, such that the nulls in each component occur at different times. In Figure 4.5c, the null in the x component is delayed 100 μs from the null in z, and 200 μs from the null in y. This is qualitatively and quantitatively similar to the effect observed in Figure 4.3c.

A close examination of power spectra of the four waveform snapshots shown in Figure 4.3 lends some support to this interpretation; however, for TRICE TAEFWD data, the frequency resolution of the spectra is limited due to the short duration of the waveform snapshots. Figure 4.6 shows power spectra with the horizontal axis zoomed in to a narrow range of frequencies centered around f_{pe} for a given snapshot. The dashed, black line shows power in the (Z') component, which is roughly parallel to B_0, while the red line shows 'perpendicular power', which is the sum of the power in the X and Y' components. As expected, more power is generally found in the parallel direction, but anywhere from 5% to 50% of the power near f_{pe} can be found in the perpendicular direction, depending on snapshot. This implies that a significant fraction of the waves present lie in the oblique regions of k-space (i.e. away

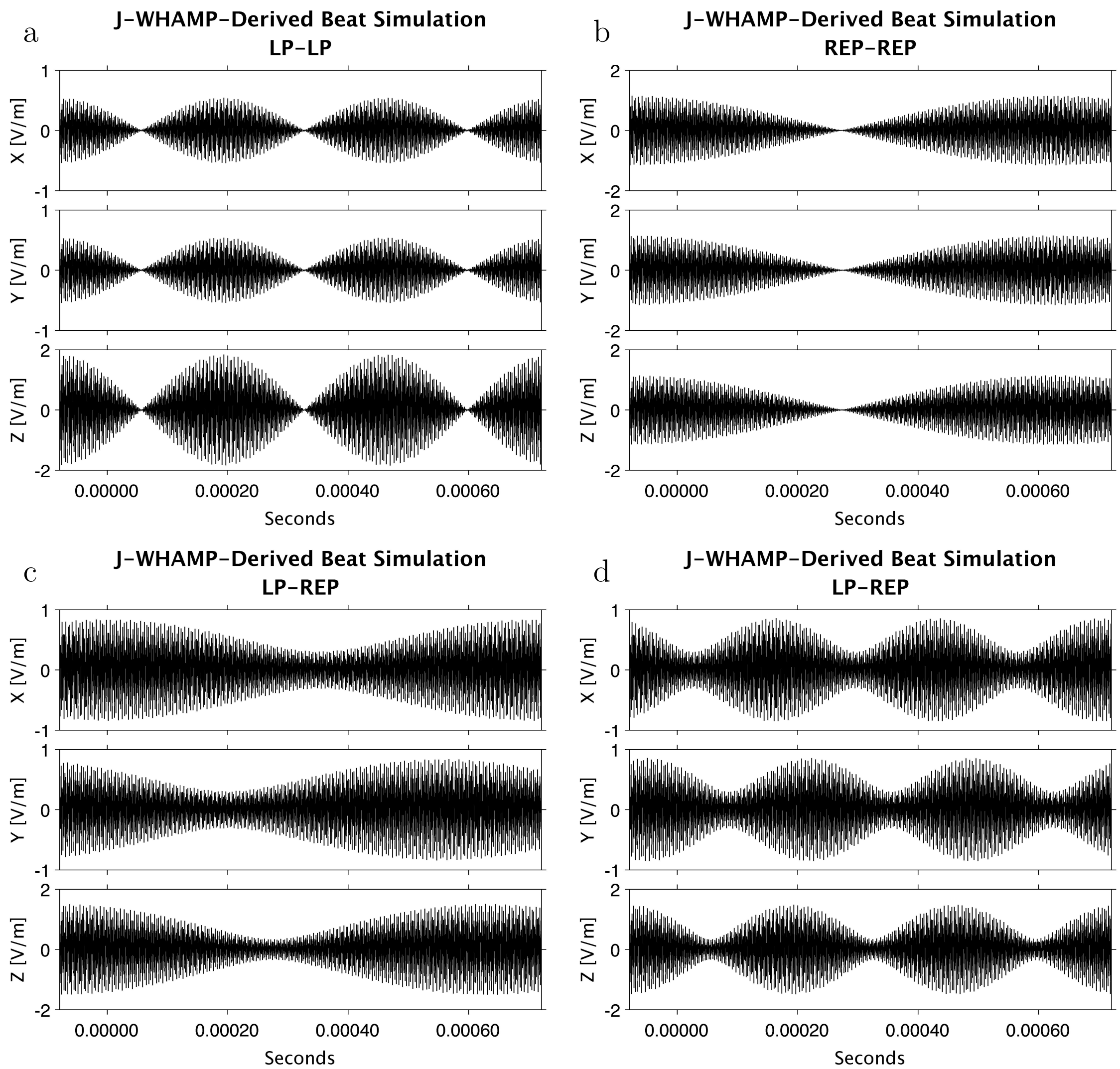

Figure 4.5: Simulations of beating between pairs of waves corresponding to Langmuir-whistler modes calculated with J-WHAMP. In **a** and **b** the simulated beating wave modes have the same polarization, both linearly polarized (**a**) or both right-elliptically polarized (**b**). In **c** and **d** one wave mode is used from the set in **a**, and one from **b**, so the simulated waves have different polarizations, one more-linear, one more-elliptical. **a** and **b** result in isotropic modulation of the three field components, whereas **c** and **d** result in anisotropies. Note that all waves have been rotated through 45 degrees so that J-WHAMP-calculated mode beating will be optimally detected (see Section 4.5).

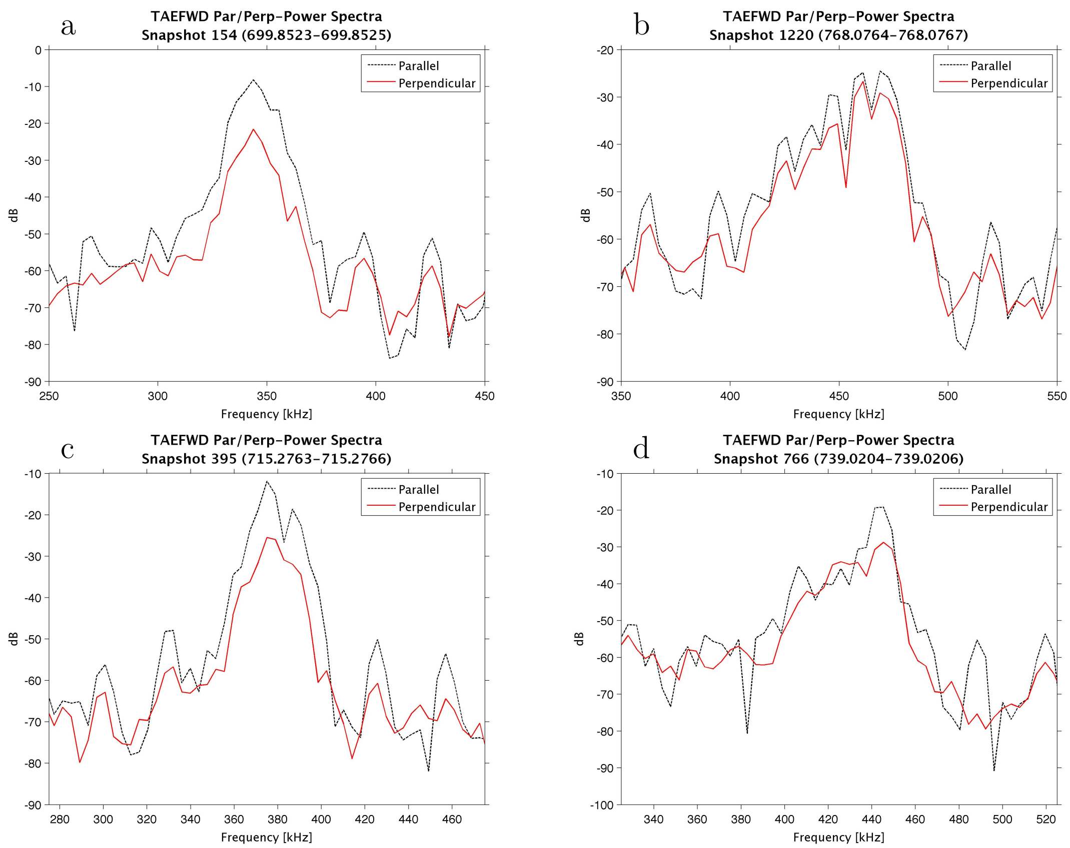

Figure 4.6: Power spectra of the TAEFWD waveform snapshots shown in Figure 4.3, showing power in the parallel (Z') and perpendicular ($X+Y'$) components. The horizontal axis shows a 200 kHz band centered on f_{pe} for the given snapshot. The significant power in the perpendicular direction implies that wave modes from a wide region of k-space are present.

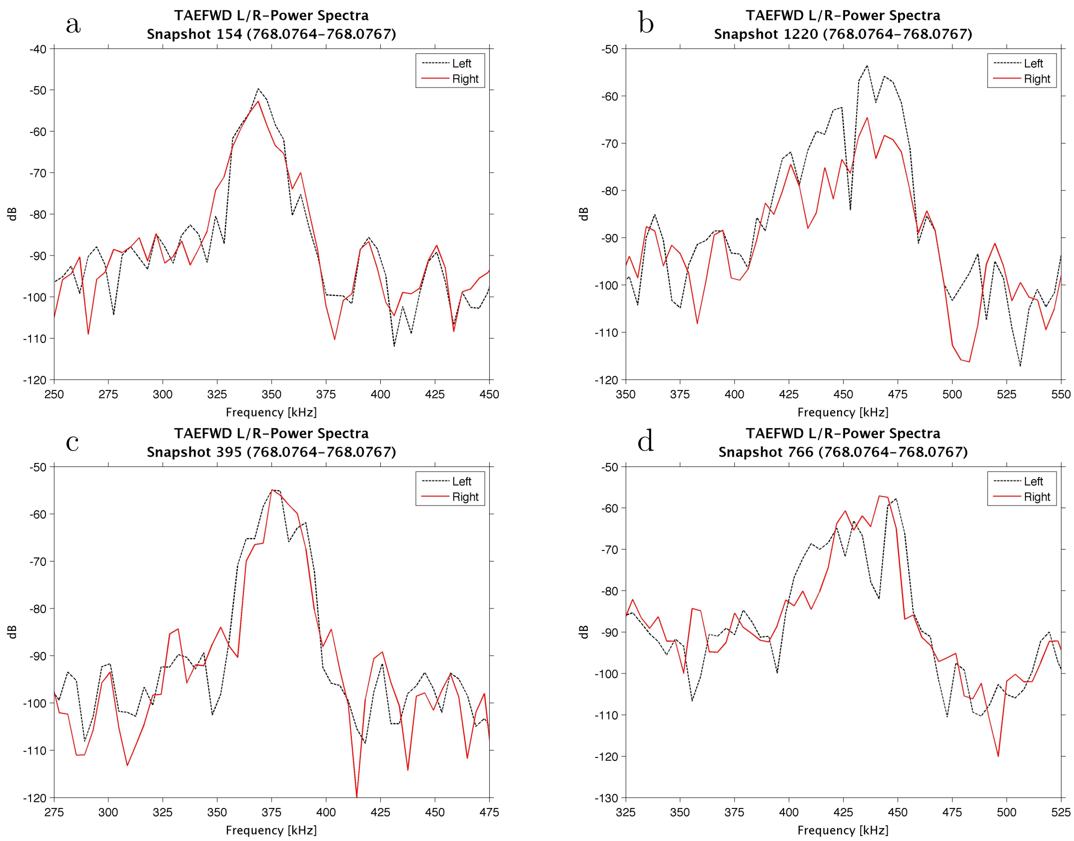

Figure 4.7: Power spectra of the TAEFWD waveform snapshots shown in Figure 4.3, with complex Fourier transforms of the transverse components (X and Y') recombined to estimate the degree of right and left circular polarization, as done in *Kodera et al.* (1977) and *LaBelle et al.* (2010). The horizontal axis shows a 200 kHz band centered on f_{pe} for the given snapshot. While the snapshots are too short to resolve the mode composition in detail, **b**, **c**, and **d** show hints that both linear and elliptical polarizations contribute to the wave modes measured near the Langmuir frequency.

from the x and y axes in Figure 4.4), which implies that multiple wave modes with different polarizations may be present.

A method of spectral analysis of polarization is taken from *LaBelle and Treumann* (1992), adapted from *Kodera et al.* (1977). Given time series data corresponding to two perpendicular, transverse components of the wave electric field, as from the measured X and reconstructed Y' components from the TAEFWD, a spectral power can be estimated for left- and right-polarized waves by recombining the complex Fast Fourier Transforms (FFT) of the time series, according to

$$\begin{aligned} FFT_L &= FFT_X + \mathrm{i}FFT_Y, \\ \text{and} \quad FFT_R &= FFT_X - \mathrm{i}FFT_Y. \end{aligned}$$

The relative power ratio $|FFT_L|^2/|FFT_R|^2$ indicates whether the waves at a given frequency are predominately left, right, or linearly polarized. Figure 4.7 shows the results of this analysis, again zoomed in near f_{pe}. While the small number of samples in each snapshot limit frequency resolution such that the individual peaks for the beating waves are not resolved, the variations seen in these spectra suggest that mixing of linear and elliptically-polarized waves is occurring to some degree.

4.5 The $k_\perp$ Ambiguity

The background magnetic field provides a natural axis in a plasma environment, which motivates a $k_\parallel$ and $k_\perp$ coordinate system, but with rotational freedom around $k_\parallel$; i.e., $k_\perp$ can lie anywhere in a plane perpendicular to the background magnetic field. J-WHAMP resolves this ambiguity by defining $k_\parallel$ to be along the z axis, $k_\perp$ to be along the x axis, and $k_y = 0$. While $\vec{k}$ does not directly affect the simulations because of the simplification that $\vec{x} = \vec{0}$, information on the wave's orientation will be a part of the complex $\vec{E}_0$, and so a more general simulation will have $k_x \neq k_y \neq 0$. One can simulate such a state by rotating one of the component waves in the beat simulations around $\hat{z}$. Looking at a general $\hat{z}$ rotation by an angle ϕ, and assuming $\vec{E}_0 \in \mathbb{R}$ so that a 100% right-circularly-polarized wave will have components $(E_x, \mathrm{i}E_y, E_z)$,

$$\begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} E_x \\ \mathrm{i}E_y \\ E_z \end{pmatrix} = \begin{pmatrix} E_x\cos\phi - \mathrm{i}E_y\sin\phi \\ E_x\sin\phi + \mathrm{i}E_y\cos\phi \\ E_z \end{pmatrix} .$$

Through judicious use of Euler identities, this can be manipulated into the form

$$\begin{pmatrix} \mathfrak{e}^{\mathrm{i}\phi} \cdot f(E_x, E_y, \phi) \\ \mathfrak{e}^{\mathrm{i}\phi} \cdot g(E_x, E_y, \phi) \\ E_z \end{pmatrix} \Bigg| (f, g) \in \mathbb{C} .$$

This shows that the rotation can be broken down into complex ϕ-dependent amplitudes f and g times an equal phase shift of the x and y components with respect to z. The fact that the x and y modulation phases remain synchronous would imply that not all of the modulation phase anisotropy seen in TAEFWD data can be explained simply by the beating waves having different wavevectors. In Figure 4.8 this is confirmed in simulation, showing the change in beat patterns while rotating one of the component waves. While there is some small shift in the x-y phase alignment, it is insufficient to reproduce, for example, Figure 4.3c, and can probably be attributed to the small difference in ellipticity between the two chosen wave modes.

An additional effect of the J-WHAMP alignment of $k_\perp$ arises because the axes in 'WHAMP-space' are effectively probes in the simulated plasma environment. With the axes aligned with the waves as output by J-WHAMP, the effect is such that any beating caused by interaction between E_x and E_y cannot be seen, as in Figure 4.9 where no modulation is seen in the y component, which is qualitatively similar to the lack of modulation in the z component of Figure 4.3d. As depicted in Figure 4.9, testing of various amounts of rotation of all beating

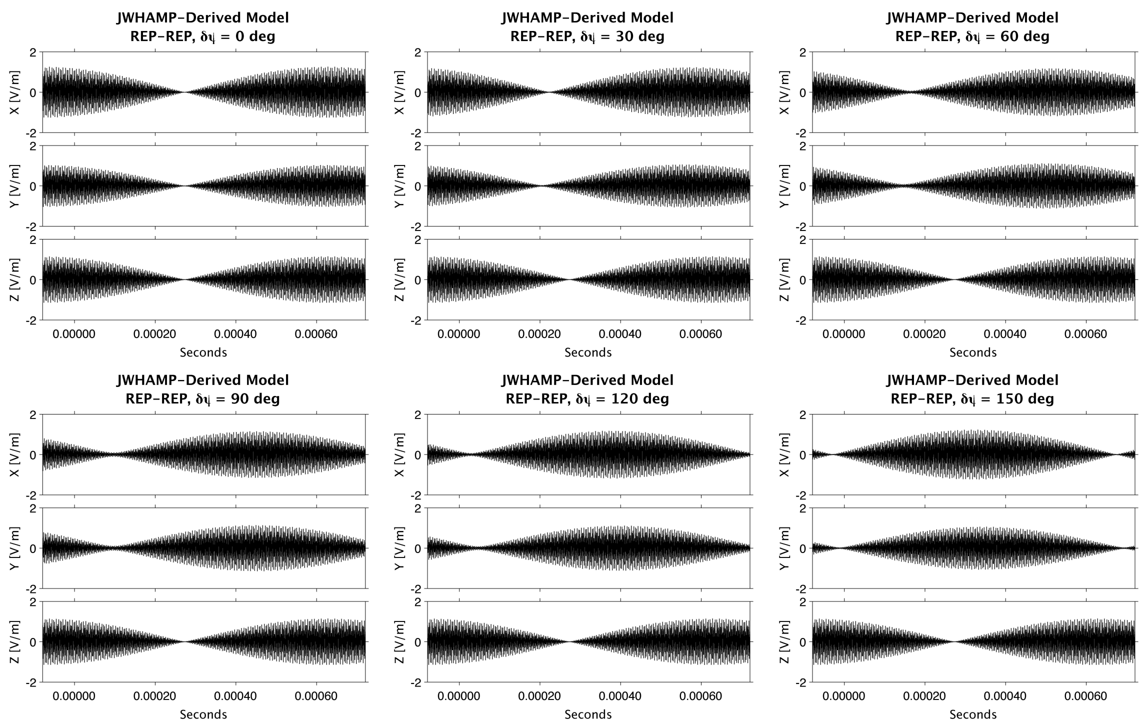

Figure 4.8: Simulation of two beating, mostly-circularly polarized waves, with one held constant and the other rotated through angle ϕ around the z axis. Note that the x and y modulation phases remain mostly aligned.

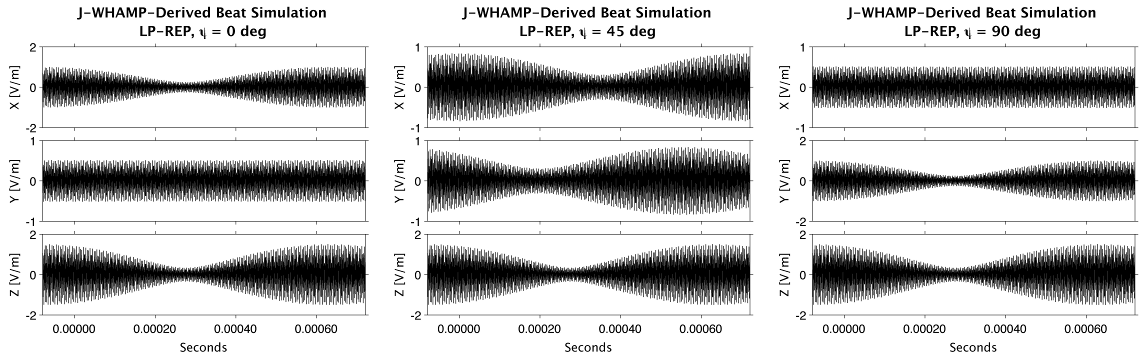

Figure 4.9: Results of rotating the simulation coordinate system through various angles ψ around the z axis, which is essentially the same as changing the orientation of the virtual rocket probes. The original orientation with $\psi = 0$ is entirely insensitive to beating in the y direction. Rotation through $\psi = 90$ degrees causes this insensitivity to move to the x direction, as one would expect with perpendicular axes. Strong beating in all channels with anisotropic modulation phase can be seen with $\psi = 45$ degrees.

component waves around the z axis by an angle ψ (effectively rotating the 'virtual payload' used to 'detect' k and $\vec{E}$), yields the conclusion that the payload orientation can have a significant impact on how well any modulation will be detected, and that $\psi = 45$ degrees is the optimal angle to detect beating in our simulations. It is possible that the physical version of such effects may be seen in the TAEFWD observations, in cases where strong modulations are seen in only two of three directions, e.g. Figure 4.3d.

4.6 Conclusions

An analysis of a period of strong Langmuir waves observed in cusp aurora by two HF electric field instruments on the TRICE high-flyer sounding rocket shows many examples of Langmuir wave bursts modulated at approximately 10 kHz. Previous studies have explained these observations as the result of beating between waves with very close frequencies near the Langmuir cutoff.

The unique 3-D data set provided by the NASA GSFC TAEFWD instrument shows that up to 25% of waveforms selected from the most-intense bursts exhibit anisotropic modulations, i.e. the beat nulls and peaks are not aligned in time across the three perpendicular electric field components. Anisotropic modulation can arise when superposed wave normal modes possess differing polarizations, e.g. if a more linearly-polarized Langmuir wave mixes with a right-elliptically-polarized whistler wave. The J-WHAMP numerical dispersion code shows that conditions appropriate to the observations can produce such waves, and simulations of such superpositions show that they do produce anisotropic modulation. Analysis of wavevector ambiguities (Section 4.5) implies that the orientation of the beating waves with respect to each other and to the instrument probes cannot fully explain the observed effect, though they can mask it. FFT analysis of the 3-D waveform data, though limited due to the short duration of waveform snapshots, suggests that both linear and elliptically-polarized waves are present near the Langmuir cutoff at these times. Because either proposed origin of the beating waves could produce waves with multiple polarizations, these findings do not resolve the origin of the multiple modes. Nevertheless, these observations illustrate how 3-D measurements can give valuable insight into the nature of wave interactions in the auroral plasma environment, and suggest that future measurements should have a higher duty cycle, and perhaps even be continuous.

Bibliography

André, M., Dispersion surfaces, *Journal of Plasma Physics*, *33*(01), 1–19, doi:10.1017/S0022377800002270, 1985.

Beghin, C., J. L. Rauch, and J. M. Bosqued, Electrostatic plasma waves and hf auroral hiss generated at low altitude, *J. Geophys. Res.*, *94*(A2), 1359–1378, 1981.

Boehm, M. H., Waves and static electric fields in the auroral acceleration region, Ph.D. thesis, University of California, Berkeley, Berkeley, CA, 1987.

Bonnell, J., P. Kintner, J. E. Wahlund, and J. A. Holtet, Modulated langmuir waves: Observations from freja and scifer, *J. Geophys. Res.*, *102*(A8), 1997.

Ergun, R. E., C. W. Carlson, J. P. McFadden, J. H. Clemmons, and M. H. Boehm, Evidence of a transverse langmuir modulational instability in a space plasma, *Geophys. Res. Lett.*, *18*, 1177, 1991.

Gurnett, D. A., J. E. Maggs, D. L. Gallagher, W. S. Kurth, and F. L. Scarf, Parametric interaction and spatial collapse of beam-driven langmuir waves in the solar wind, *J. Geophys. Res.*, *86*(A10), 8833–8841, 1981.

Kaeppler, S., C. A. Kletzing, S. R. Bounds, J. W. Gjerloev, B. J. Anderson, H. Korth, J. W. LaBelle, M. P. Dombrowski, M. Lessard, R. F. Pfaff, D. E. Rowland, S. Jones, and C. J. Heinselman, Current closure in the auroral ionosphere: Results from the auroral current and electrodynamics structure rocket mission, submitted Nov 2011, 2011.

Khotyaintsev, Y., G. Lizunov, and K. Stasiewicz, Langmuir wave structures registered by freja: Analysis and modeling, *Adv. Space Res.*, *28*, 1649, 2001.

Kintner, P. M., J. Bonnell, R. Arnoldy, K. Lynch, C. Pollock, T. Moore, J. Holtet, C. Deehr, H. Steinbaek-Nielsen, R. Smith, and J. Olson, The scifer experiment, *J. Geophys. Res.*, *101*, 1865, 1996.

Kletzing, C. A., J. LaBelle, D. E. Rowland, R. F. Pfaff, S. Kaeppler, and M. P. Dombrowski, Untitled, in process paper on CHARM II WPI data, 2012.

Kodera, K., R. Gendrin, and C. de Villedary, Complex representation of a polarized signal and its application to the analysis of ulf waves, *J. Geophys. Res.*, *82*(7), 1245–1255, 1977.

LaBelle, J., and R. A. Treumann, Poynting vector measurements of electromagnetic ion cyclotron waves in the plasmasphere, *J. Geophys. Res.*, *97 (A9)*(13), 789–797, 1992.

LaBelle, J., I. H. Cairns, and C. A. Kletzing, Electric field statistics and modulation characteristics of bursty langmuir waves observed in the cusp, *J. Geophys. Res.*, *115*(A1), 317, 2010.

Layden, A., I. H. Cairns, P. Robinson, and J. LaBelle, Changes in mode properties versus mode conversion for waves in earth's auroral ionosphere, *J. Geophys. Res.*, *116*(A12328), 2011.

Lizunov, G. V., Y. Khotyaintsev, and K. Stasiewicz, Parametric decay to lower hybrid waves as a source of modulated langmuir waves in the topside ionosphere, *J. Geophys. Res.*, *106*, 24,755, 2001.

Maggs, J. E., Coherent generation of vlf hiss, *J. Geophys. Res.*, *81*, 1707, 1976.

Malaspina, D. M., and R. E. Ergun, Observations of three-dimensional langmuir wave structure, *J. Geophys. Res.*, *113*(A12108), 2008.

McAdams, K. L., J. LaBelle, P. W. Schuck, and P. M. Kintner, Phaze ii observations of lower hybrid burst structures occurring on density gradients, *Geophys. Res. Lett.*, *25*(16), 3091–3094, 1998.

McAdams, K. L., J. LaBelle, and M. L. Trimpi, Rocket observations of banded structure in waves near the langmuir frequency in the auroral ionosphere, *J. Geophys. Res.*, *104*(A12), 28,109–28,122, 1999.

McFadden, J. P., C. W. Carlson, and M. H. Boehm, High-frequency waves generated by auroral electrons, *J. Geophys. Res.*, *91*, 12,079, 1986.

Newman, D., M. Goldman, and R. E. Ergun, Langmuir turbulence in the auroral ionosphere, 2. nonlinear theory and simulations, *J. Geophys. Res.*, *99*, 6377, 1994a.

Newman, D., M. Goldman, R. E. Ergun, and M. H. Boehm, Langmuir turbulence in the auroral ionosphere, 1. linear theory, *J. Geophys. Res.*, *99*, 6367, 1994b.

Rönnmark, K., Whamp- waves in homogeneous, anisotropic multicomponent plasmas, *Tech. Rep. 179*, Kiruna Geophysical Institute, 1982.

Samara, M., and J. LaBelle, Lf/mf whistler mode dispersive signals observed with rocket-borne instruments in the auroral downward current region, *J. Geophys. Res.*, *111*(A09305), 2006.

Samara, M., J. LaBelle, C. A. Kletzing, and S. R. Bounds, Rocket observations of structured upper hybrid waves at fuh = 2fce, *Geophys. Res. Lett.*, *31*(22), 2004.

Stasiewicz, K., B. Holback, V. Krasnoselskikh, M. H. Boehm, R. Boström, and P. M. Kintner, Parametric instabilities of langmuir waves observed by freja, *J. Geophys. Res.*, *101*(21), 515, 1996.

Chapter 5

Coda

Waves are pervasive in the auroral ionosphere. Langmuir waves in particular are ubiquitous, and a carrier of energy from their source regions in the magnetosphere, through and into the ionosphere. They can be useful as probes of ionospheric density, and for remote sensing. To fully understand Langmuir waves and make use of them, a thorough understanding of their generation, propagation, damping, and interactions with particles and other waves is necessary, and as Langmuir waves are generated in many other space plasma and laboratory environments, such results can be widely applicable. Several steps have been taken towards improving knowledge of and theories regarding Langmuir waves.

An autonomous, high-speed, digital signal-processing receiver has been developed and refined. The Dartmouth Rx-DSP is a flexible tool, capable of observing fine frequency-time structures, polarization, and source direction, and of onboard data reduction. The features of the Rx-DSP make ARCs built from them ideal for multifaceted studies of Langmuir waves and related phenomena, both remotely and in-situ. ARCs using these receivers have already made new science observations in both ground and sounding rocket deployments, including the first in-situ fine-structure observation of the polarization of auroral roar.

Wave-particle Correlator data from the CHARM-II mission has been presented and analyzed. This constitutes the first statistical observation from a Langmuir-wave Correlator system, and a resistive/reactive fit of the data shows a direct relation between the fit coefficients and electron-beam onset and dissipation. This relation implies that, for a beam which shows an increased Landau-resonance growth rate during onset, beam dissipation will show enhanced damping as the beam dissipates. The data also shows comparable levels of resistive and reactive activity, despite the high electric-field strength which should cause very swift relaxation of resistive electron populations. A flexible, numerical test-particle simulation has been developed to test the plausibility of this conclusion, and simulated results appear to be qualitatively in agreement with theories explaining these observations.

Finally, data from the unique, three-dimensional, high-frequency TAEFWD instrument flown on the TRICE mission has been presented. This data has allowed an unique examination of ionospheric bursty Langmuir waves in the cusp, which are theorized to result from wave-wave interaction. Comparison of the data to simulations of beating waves and results from

J-WHAMP imply that the interacting waves are some mixture of pure, linearly polarized Langmuir waves, and elliptically polarized, partially oblique modes from the upper-hybrid dispersion surface, commonly referred to as whistler-Langmuir hybrid waves.

5.1 Future Work

Future iterations of the Rx-DSP platform may include both hardware and software improvements. The Rx-DSP shows great potential, and has already proven itself as an effective tool, but a number of design flaws are included in the current version of the hardware—see Chapter A—which need to be addressed in future iterations; as well, newer versions of the core processors could increase the platform's basic capabilities, such as digitization bit-depth and DSP RAM. Potential firmware improvements include lossless or lossy data compression, live signal detection and center-frequency tracking, and further enhancements of ARC autonomy.

Concurrent with analysis and publication of the TRICE results, significant three-dimensional wave data from the STEREO spacecraft have been reported on (*Malaspina and Ergun* 2008). These have lead to interpretations of bursty Langmuir-wave structure in the solar wind as eigenmodes of density cavities. Strong perpendicular fields were seen in half of a set of 732 events, implying Langmuir/z-mode waves should generally play a large part in wave decay processes (*Graham and Cairns* 2014).

A significant limiter of the high-frequency, three-dimensional measurements afforded by the TAEFWD receiver as it was flown on the TRICE mission is the single temporospatial observation point. April 2013 discussions with Dr. Konrad Sauer elucidated that the TRICE TAEFWD observations are consistent with the appearance of whistler-Langmuir soliton structures, also known as 'oscillatons' (*Sauer and Sydora* 2001). Confirmation of this and further detangling of the structure of bursty Langmuir waves in both time and space would require multiple simultaneous and synchronous TAEFWD-like observations, with a range of separations from tens to hundreds of meters. Such instrumentation could also yield new data on generative electron-beam cross-sections, and yield new results on wave propagation and the non-Langmuir participants in wave-wave interactions.

While the wave-particle Correlator is an effective system as-is, some improvements towards increasing the high-quality event/mission detection rate are possible. While increased effort was made during CHARM-II mission integration towards reducing the total payload noise as seen by the Dartmouth HFE, even more noise reduction, including through the entire Correlator system, could reduce the interference that led to many manual event screenings. In addition, while the PLL system is generally effective, some form of additional pre-filtering or digital processing of the incoming HFE signal might allow for better tracking of the Langmuir frequency, yielding even more event confidence. An additional study which could be undertaken within the CHARM-II dataset is a search for further relations to the resistive and reactive components; e.g., while CHARM-II did not see large numbers of bursty Langmuir waves, a detailed manual examination of frequency splitting and spacings could prove edifying.

Both the numerical magneto-kinetic test-particle simulation and the growth rate calculation codes developed for comparison with Correlator results are extremely flexible tools. The test-particle simulation is easily expanded to different environments and parameter-spaces, including the potential to add a background electric field, and to vary travel distances, field strengths, field shapes, and particle charges and masses. The distributed, node-independent nature of the test-particle code allows for division of work among as many nodes and cores as are available, optimizing simulation run times given available resources. The distribution builder and growth rate calculator are modular, allowing for easy input of alternate top-side distribution functions with angular dependencies, and for arbitrary time-varying beam distributions.

An improvement which is desirable but not immediately attainable is for higher time-resolution growth-rate calculations. The time-overlapping nature of the distribution function data—as well as the requirement $\Delta t_D/\Delta t_S > 10$—swiftly leads to RAM and CPU requirements becoming untenable when attempting to push the simulated detector cadence down towards realistic millisecond values. Further coding effort would be required in order to make a cluster-deployable version of the growth-rate code stack, enabling detailed examinations of the millisecond-scale growth rate reactivity.

Characteristics of bursty Langmuir waves in the solar wind and Earth's foreshock have been quantitatively explained via Stochastic Growth Theory (*Cairns and Robinson* 1997; *Cairns et al.* 2000; *Boshuizen et al.* 2001). Key to this theory is the effects of small-scale density inhomogeneities in the plasma. While the current simulation could theoretically model an inhomogeneous source region, numerical testing which includes density irregularities during electron beam transit would require a simulation such as a Particle-in-Cell code.

There are some quickly accessible future realms of study available, making use these codes with no or minimal revision, even using the existing set of test-particle data. These include highly dynamic scenarios with multiple beams at multiple energies, beams with time-varying source energies or limited angular extents, and even beams with varying azimuthal dependence. When the spatial component of the test particle simulation was discarded towards the end of the test-particle simulation, it equated to an implicit assumption of source-region homogeneity, but this is known not to be unrealistic. As positions are present and accurate in the data, they could be utilized for studies involving the spatial extent of Langmuir-wave generating beams. Augmentations to the growth-rate code stack would allow for examinations of growth rates for obliquely propagating waves. Finally, the test-particle simulation is capable of runs with background electric fields, higher or lower particle launch energies, and higher resolution in both energy and pitch angle.

A more accurate simulation of the situation is possible, and as shown any number of the above factors may contribute to the limiting of any quantitative conclusions. As binning has been shown to affect the small-timescale growth rate behavior, additional particle runs to 'fill in' between the current launch energies may allow for reaching smooth, stable, and more realistic results. Efficiency improvements and cluster capabilities would also allow for more realistic beam lifetimes, and potentially probing of behavior at timescales the Correlators can not yet work at.

Bibliography

Boshuizen, C. R., I. H. Cairns, and P. A. Robinson, Stochastic growth theory of spatially-averaged distributions of langmuir fields in earth's foreshock, *Geophys. Res. Lett.*, *28*(18), 3569–3572, doi:10.1029/2000GL012709, 2001.

Cairns, I. H., and P. A. Robinson, First test of stochastic growth theory for langmuir waves in earth's foreshock, *Geophys. Res. Lett.*, *24*(4), 369–372, doi:10.1029/97GL00084, 1997.

Cairns, I. H., P. A. Robinson, and R. R. Anderson, Thermal and driven stochastic growth of langmuir waves in the solar wind and earth's foreshock, *Geophys. Res. Lett.*, *27*(1), 61–64, doi:10.1029/1999GL010717, 2000.

Graham, D. B., and I. H. Cairns, Dynamical evidence for nonlinear langmuir wave processes in type iii solar radio bursts, *J. Geophys. Res.*, *119*(4), 2430–2457, doi:10.1002/2013JA019425, 2014.

Malaspina, D. M., and R. E. Ergun, Observations of three-dimensional langmuir wave structure, *J. Geophys. Res.*, *113*(A12108), 2008.

Sauer, K., and R. D. Sydora, Whistler-langmuir oscillitons and their relation to auroral hiss, *Annales Geophysicae*, *29*(10), 2001.

Appendices

Appendix A

Rx-DSP Notes & Code

Documenting the Rx-DSP codes is a bit of a gong show. The code is slightly modified for every deployment, there's kludges nobody even remembers, and in some cases the code is copyright Texas Instruments, so no bueno. Also, assembly code is absurdly long—just the AGO code is over 5,000 lines. The full codebase is available at the group's GitHub repository, https://github.com/DartmouthSpacePhys.

The following will, instead, attempt to go over the history of the code, the basic parts of the modern structure, and compilation, and the parts of the code which are Dartmouth-specific and have unique features. It will also cover a couple of 'quirks' that have been found in the design.

A.1 Original Rocket Code & Errata

The original code still used on rockets was written by J. C. Vandiver, based on Iowa code for the hardware the Rx-DSPs themselves were based on. It is a single, monolithic code block that contains the serial monitor code, AD6620 RSP support code, and the main data acquisition code. It takes data in 65 kiloword frames, triggered by a Major Frame Interrupt from the telemetry hardware, and is designed to be used in a two-DSP synchronized setup, in order to get polarization data. It synchronizes the major frames the two DSPs are sending by using the FIFO-reset line wired between, and herein lies an issue.

Design Flaw: FIFO Reset

The FIFO Reset line, as designed, has a flaw when you try to use it as above, while the system is 'live' (i.e. the RSP is running and the FIFO is filling with data).

The base problem is that no part of the system pays attention to where the RSP is in its data cycle, or what data has been pulled off of it by the FIFO. If you're watching the RSP output, the duty cycle is such that it spends a large time idle, then sends the I word of a sample, and then the Q word a short—but non-zero—amount of time later. If the FIFO

Reset line is triggered in that tiny time gap between I and Q, the first word loaded into the now-empty FIFO will be a Q.

The system is designed to account for this using the least significant bit in the first few words of data in each block. There is hardware which is toggled on by the c542 which watches the I/Q line on the RSP output, and sets the bit high or low depending on its I or Q state. In the rocket code, this hardware is set on for the first 200 words of each major frame. In theory it should thus be possible to determine the 'I/Q phase' of a major frame by examining these LSBs.

Unfortunately, this LSB-override hardware is unreliable: it has been observed on the bench to yield incorrect results in a small number of cases, for reasons unknown. A possible more-robust option would be to add hardware which holds the FIFO Reset line high, if the I/Q line is in Q state, for as long as it remains so, thus preventing a Q from ever being read in as the first word.

There are two workarounds, one in post-processing, and the other in firmware operation structure. The post-processing workaround requires that a well-known signal be present in the data, e.g. a beacon or interference line. Wit this the the data can be closely perused, and any major frames which appear to be 'bad' (i.e. discontinuous with respect to neighboring frames) can be tagged as such, and processing attempted with the first word dropped.

The firmware workaround is to change the way the RSP is handled. The only state in which you are guaranteed to start with an I is the startup state, so if you send the RSP into reset at the end of each major frame, then start it fresh just before a new frame, the problem is solved...right? Well, sort of. You also have to discard a chunk of data at the very start—the first few words put out are nonsense, and the filters need a while to kick in. For safety, current codes discard the first 512 words out of the FIFO.

An added benefit of the firmware workaround is that it slightly lowers the power consumption, depending on what fraction of time the RSP spends in the idle state.

The rocket firmware is very barebones, simply reading in a major frame, adding a header, and then putting everything on the output FIFO for telemetry hardware to read out.

Design Quirk: Receive FIFO Clocking Possibly because of its history as a University of Iowa instrument, the receive FIFO hardware is designed such that it is directly clocked by the c542 read action—this makes the FIFO status lines unreliable, and requires that the firmware data-read loop manually time its read operations with `nop`.

While perfectly valid, this method ties the c542 up for an inordinate amount of time. All revised deployments of the Rx-DSP have been hardware-modified to have a more traditional setup, with the FIFO clock line receiving the onboard 40 MHz clock signal. This allows an idle state or other work to take place, until the FIFO status lines reach a state which requires/allows for read-out.

The rocket code contains several parts: interrupt vectors, definition of constants, serial monitor code and utilities, the RSP programming code, and the main acquisition program.

Some of these shall be covered in a general sense by the following sections, which review updated versions.

A.2 South Pole Station PF-ARC

The receiver at South Pole Station was the first ground-based deployment of the Rx-DSP, and is very close to a rocket setup, using an essentially unmodified firmware. The unique part of the setup lies on the 'telemetry' side: each of the Rx-DSPs has its parallel high-speed output port wired to a Bitwise Systems QuickUSB module, and both modules are connected to a standard PC running Linux.

The QuickUSB is wired to directly trigger major frames, acting similarly to a rocket telemetry system to pull data from the Tx FIFO. The acquisition code was custom written with assistance from Bitwise, and is available on the repository.

A.3 The Antarctic AGO S-ARC

This section covers codes specifically for an S-ARC designed for use at Antarctic Automatic Geophysical Observatory (AGO) sites. This design takes 512-sample bursts of data, then FFTs the data, and transmits the power in decibels through the c542 buffered serial port.

While these codes are purpose-specific, the general forms are the same as for any single-receiver ARC.

A.3.1 Compilation Support/Toolchain

The codes are compiled and linked from `.asm` to `.obj`, and then linked into a single loadable `.hex`, using a suite of Texas Instruments command line programs. Below are three support files for this process: a GNU `Makefile`, two `.cmd` files, which are used by the `hex500.exe` and `lnk500.exe` programs as sources of additional options, and a short bootloader code.

The files below are designed for a two-stage initialization process, which uses a bootloader code for a more recent DSP, the c549. This bootloader allows more advanced memory management—specifically, having the program code split between multiple memory regions—which is required to fit the program codes and lookup tables into RAM, leaving enough room for data storage, FFTing, and transmission.

The lookup tables, which take over 2 kilowords of storage alone, store pre-calculated values of sine functions for the Hann windowing and FFT functions.

`Makefile`

```
MAINSRC ?= ago_v1.0.asm
CFPREFIX ?= twostage
```

```
ASMFLAG = -v542
LNKFLAG = -w -a -r

CGTDIR = /cygdrive/c/TI54xCGT/bin

BASESRC = int_table.asm bl549.asm ad6620.asm tablemax.asm
FUNCSRC = $(BASESRC) Cbrev32.asm c512.asm log_10.asm hannwin.asm sercook.asm cfft_32.asm dpsm.asm scale.asm
TABLES = hann_q15.tab

OBJECTS = $(MAINSRC:.asm=.obj) $(FUNCSRC:.asm=.obj) $(TABLES:.tab=.obj)
LSTS = $(FUNCSRC:.asm=.lst) $(MAINSRC:.asm=.lst)
ABSS = $(FUNCSRC:.asm=.abs) $(MAINSRC:.asm=.abs)

OUTMAP = $(MAINSRC:.asm=-link.map)
HEXMAP = $(MAINSRC:.asm=-hex.map)

OUTFILE = $(MAINSRC:.asm=.out)
HEXFILE = $(MAINSRC:.asm=.hex)

.SUFFIXES: .asm .obj .abs .lst .hex .out .tab

all: $(HEXFILE) $(LSTS)

$(HEXFILE): $(OUTFILE) $(CFPREFIX)hex.cmd
	$(CGTDIR)/hex500.exe $(HEXFLAG) $(CFPREFIX)hex.cmd -map $(HEXMAP) -o $(HEXFILE) -i $(OUTFILE)

.abs.lst: $(ABSS)
	$(CGTDIR)/asm500.exe $(ASMFLAG) -x -a $<

.obj.abs: $(OUTFILE)
	$(CGTDIR)/abs500.exe $(OUTFILE)

$(OUTFILE): $(OBJECTS) $(CFPREFIX)link.cmd rx-dsp.h int_table.h ad6620.asm
	$(CGTDIR)/lnk500.exe $(LNKFLAG) $(CFPREFIX)link.cmd -m $(OUTMAP) -o $(OUTFILE) $(OBJECTS)

.tab.obj:
	$(CGTDIR)/asm500.exe $(ASMFLAG) $< $@

.asm.obj:
	$(CGTDIR)/asm500.exe $(ASMFLAG) $< $@

clean:
	rm -f $(OBJECTS) $(LSTS) $(ABSS) $(OUTFILE) $(HEXFILE) $(OUTMAP)
```

```
        $(HEXMAP)
```

twostagehex.cmd

```
/* TMS320C542 DSP Board Boot Rom Generation Command File */
/* 19 Dec. 2009- updated for c: drive */
/* 01 Nov. 2006 */
/* Dartmouth MASTER Rx-DSP Boot PROM Generation */

-memwidth 8
-romwidth 8
-boot             /* Convert all COFF sections to hex */
-bootorg 0x0000        /* External data memory boot */
-swwsr 0x7FFF

ROMS {
    EPROM1: origin=0x0000, length=0x8000, memwidth=8, romwidth=8
}

/*SECTIONS {
    .bl549 = boot
    .vectors
    .cbrev_p
    .cfft_p
    .log10_p
    .smon_p
    .sintab
}*/
```

twostagelink.cmd

```
/* TMS320C542 DSP Board Boot Rom Linker Command File */

-e RXDSP_START

MEMORY {
PAGE 0:
    INTR_TABLE (RWX): origin = 0x0080, length = 0x0080
    PROG_ANNEX (RIX): origin = 0x0180, length = 0x0680
    PROG_MAIN (RIX) : origin = 0x0C00, length = 0x1200

PAGE 1:
    STACK (RW)      : origin = 0x0100, length = 0x0040
    TEMP_DATA (RW)  : origin = 0x0140, length = 0x0040
    SBUFFER (RW)    : origin = 0x0800, length = 0x0400
    SCALES (RW)     : origin = 0x1E00, length = 0x0200
    DATA (RW)       : origin = 0x2000, length = 0x0800
```

```
}

SECTIONS {
    .bl549        : load > PROG_ANNEX, align = 64
    .vectors      : load > INTR_TABLE
    .text         : load > (RIX)
    .cbrev_p      : load > (RIX)
    .cfft_p       : load > (RIX)
    .log10_p      : load > (RIX)
    .smon_p       : load > (RIX)
    .smon_msg     : load > (RIX)
    .sine_tab     : load > (RI)
    .hann_tab     : load > (RI)
    .hann_p       : load > (RIX)
    .sercook_p    : load > (RIX)
    .ad6620       : load > (RIX)
    .transfer_p   : load > (RIX)
    .dpsm_p       : load > (RIX)
    .scale_p      : load > (RIX)

    /* data sections */
    .bss        : > TEMP_DATA
    .stack_v    : > STACK
    .sbuff_v    : > SBUFFER
    .scale_v    : > SCALES
    .data_v     : > DATA
}
```

bl549.asm

```
;************************************************************
;*** Bootloader software version NO. : 1.0 ***
;*** Last revision date : 10/23/1996 ***
;*** Author : J. Chyan ***
;************************************************************
;** **
;** Boot Loader Program **
;** **
;** This code segment sets up and executes boot loader **
;** code based upon data saved in data memory **
;** **
;** WRITTEN BY: Jason Chyan **
;** DATE: 06/06/96 **
;** **
;** Revision History Omitted **
;************************************************************

```

```
;*************************************************************
;.title ""bootc54LP
;***********************************************************
; symbol definitions
;***********************************************************
.mnolist

; Let's use some scratchpad memory!  Woo!
brs       .set    60h ; boot routine select (configuration word)
xentry    .set    61h ; XPC of entry point
entry     .set    62h ; entry point
hbyte     .set    63h ; high byte of -8bit serial word
p8word    .set    64h ; concatenator for -8bit memory load
src       .set    65h ; source address
dest      .set    66h ; destination address (dmov from above)
lngth     .set    67h ; code length
temp0     .set    68h ; temporary register0
temp1     .set    69h ; temporary register1
temp2     .set    6ah ; temporary register2
temp3     .set    6bh ; temporary register3
nmintv    .set    6ch ; -nonmaskable interrupt vector
sp_ifr    .set    6dh ; SP IFR temp reg
; MMR definition for c54xlp CPU register
;**
ifr      .set     01h
st0      .set     06h
st1      .set     07h
AL       .set     08h
AH       .set     09h
AG       .set     0Ah
brc      .set     1ah
pmst     .set     1dh
swwsr    .set     28h
bscr     .set     29h


;* * * * * * * * * * * * * * * * * * * * * * * * * *
;*    Bootload from -8bit memory, MS byte first    *
;* * * * * * * * * * * * * * * * * * * * * * * * * *

    .global    BOOTLOAD_START, blskipskip, xfr08, par08_1, endboot
    .ref       RXDSP_START
    .sect      ".bl549"
entry_point      .set    RXDSP_START
eprom_base       .set    0x8000
bl_loadpoint     .set    BOOTLOAD_START

```

```
BOOTLOAD_START
par08
    stm    #0x7FFF,swwsr    ; set full wait states
    stm    #0x0002,bscr     ; bus holder enabled
    ld     #0, DP
    nop
    nop

    st     #entry_point, @entry

    stm    #eprom_base, AR1

par08_1  ; Main section load loop

    nop
    ld     *ar1+, 8, a  ; get address of destination
    and    #0ff00h,a    ; force AG, AH to zero for correct calculation
                        ; of the -23bit destination address. (10/14/99 BCT)
    mvdk    *ar1+, ar3     ; ar3    <-- junkbyte.low byte
    andm    #0ffh, @ar3    ; ar3    <-- low byte
    or      @ar3, a        ; acc A  <-- high byte.low byte
    stlm    a,ar2          ; ar2    <-- destination address

    bc    endboot,aeq  ; section dest = 0 indicates boot end

    ld      *ar1+, 8, a  ; get number of 16-bit words
    and     #0xFF00,a    ; Clear the guard bits and keep low accum (1.92)
    mvdk    *ar1+, ar3   ; ar3    <-- junkbyte.low byte
    andm    #0ffh, @ar3  ; ar3    <-- low byte
    or      @ar3, a      ; acc A  <-- high byte.low byte

    cmpm    AR2, #bl_loadpoint  ; check if our dest is the bootloader load address
    bc      blskipskip, NTC     ; if not, keep loading

    add     #1, A   ; if it is the bootloader, we want to skip
    stlm    A, AR0  ; this section, i.e. skip A+1 words
    nop
    add     AR0, A  ; but wait,
    stlm    A, AR0  ; A+1 words = 2(A+1) addresses (8-bit prom)
    nop
    bd      par08_1
    mar     *AR1+0
    nop

blskipskip:
```

```
    stlm    a, brc     ; update block repeat counter register
    nop
    rptb    xfr08 - 1  ; block repeat to load section data

    ; load program code word
    ld      *ar1+, 8, a  ; acc A  <-- high byte
    and     #0xFF00, a
    mvdk    *ar1+, ar3   ; ar3    <-- junkbyte.low byte
    andm    #0ffh, @ar3  ; ar3    <-- low byte
    or      @ar3, a      ; acc A  <-- high byte.low byte
    stl     a, @p8word

    ; recover destination address, pause, then write and increment
    ldu      @ar2, a
    nop
    nop
    writa    @p8word
    add      #1, a
    stlm     a, ar2

xfr08:  ; end block repeat

    b     par08_1  ; end section loop

;**
;*    End 549 8-bit EPROM bootloader
;**

endboot
    ldu    @entry, a  ; branch to the entry point
    nop
    nop
    baccd  a
    nop
    nop
```

A.3.2 AD6620 RSP Support Code

The code in this file contains functions which initialize, stop (reset), and start the RSP. Not included are the filter tables which are loaded into memory, though the format is described in a comment.

`ad6620.asm`

~~~
;```````
; AD6620 setup functions and tables
~~~

```
;_______

    .mmregs
    .def      rsp_clear, rsp_reset, rsp_init, rsp_mstart, rsp_sstart
    .include  "rx-dsp.h"
    .sect     ".ad6620"

;
; rsp_reset, rsp_init, rsp_mstart, rsp_sstart
;     shell functions over rsp_setup
;

rsp_reset:
    ld      #ad6620_soft_reset, A   ; Put AD6620 into reset
    call    rsp_setup

    retd
    nop
    nop

rsp_init:
    ld      #ad6620_filter, A   ; Set up AD6620 filter
    call    rsp_setup

    retd
    nop
    nop

rsp_mstart:
    ld      #ad6620_master_run, A   ; Start digitizing as master
    call    rsp_setup

    retd
    nop
    nop

rsp_sstart:
    ld      #ad6620_slave_run, A   ; Start digitizing as slave
    call    rsp_setup

    retd
    nop
    nop

;
; rsp_clear    Function to clear RCF Data RAM between frames
;
```

```

rsp_clear:
    stm      #100000001b, AR3
    nop
    nop
    portw    AR3, (wr_rx+amr)   ; Load high and low address registers:
    stm      #0, AR3
    nop
    nop
    portw    AR3, (wr_rx+lar)   ; write to 0x100, auto-increment

    stm      #0xFF-1, BRC
    nop
    nop
    rptb     rsp_clear_loop - 1

    portw    AR3, (wr_rx+dr4)
    portw    AR3, (wr_rx+dr3)
    portw    AR3, (wr_rx+dr2)
    portw    AR3, (wr_rx+dr1)
    portw    AR3, (wr_rx+dr0)
    nop

rsp_clear_loop:

    retd
    nop
    nop

; Load RSP (AD6620) Registers from Table
;
; 23 Dec 2009 took out most writes to terminal (msgout, dis4hex, asx)
;
; This code was taken directly from the "rspmod" routine used in the
; Dartmouth Monitor. Instead of having the user enter the data words
; or receiving them from a "script" file, this routine looks at a
; table of words in memory, reads them, and transfers them to the
; AD6620. Used to load control bytes and filter coefficients.
;
; Table entry format:
;
; rsp_table:
;     .word    AmLah, r4r3h, r2r1h, r0xxh;
;     .word    (more 4-word entries)
;     .word    0FFFFh    ; End of table
;
; 4 words:
```

```
; AAaah = AD6620 internal address, 0000h to 030Dh,
;     or FFFFh to terminate.
; AMR = Ma Address mode register
; LAR = La Lower address register
; r4r3h, r2r1h, r0xxh = data bytes, packed into words, MS, to LS.
;     Bottom byte of 3rd word not used (xx). Data is treated a 40
;     bits for all AD6620 registers. Not the most compact
;     arrangement for storage, but readable- and it can be edited
;     directly from monitor scripts.
;
; DR4 = r4
; DR3 = r3
; DR2 = r2
; DR1 = r1
; DR0 = r0
;
; Uses:
; A:   Holds table pointer upon entry
; B:   Working register
; AR0: I/O address
; AR2: Table index
; AR3: Holds data to send to or read from I/O port

table_end .set 0FFFFh        ; End-of-table definition

rsp_setup:             ; Enter with table starting address in A
     stlm    A,AR2    ; Save to AR2 for later use
     nop              ; Necessary for loop to execute properly (?!&)
     nop              ; Necessary for loop to execute properly (?!&)
;    ld      #ad6620_msg1,A  ; Tell operator what is happening
;    call    msgout

rsp_loop:
;    ldm     AR2,A     ; Retrieve table index
;    call    dis4hex   ; Display index of table line
;    ld      #0020h,A  ; Space over on screen
;    call    asx

    cmpm    *AR2,#table_end  ; Is this the end of the table?
    bc    rspx,TC            ; Return if at end

    ld    *AR2+,A  ; Get first table word: AD6620 address
    ld    A,B      ; Save a copy

;    call    dis4hex  ; Display

;
```

```
; Transfer RSP register address bytes to
; AD6620 high and low address registers
;
    sftl    A,-8,A       ; Shift high byte to low byte
    and     #0003h,A,A   ; Mask high byte to 2 LSBs (avoid reserved bits
                         ; and do not auto-increment for now)
    stlm     A,AR3           ; Move to AR3 for portw
    portw    AR3,wr_rx+amr  ; Write to high address register
    ld       B,A          ; Get RSP register address
    and      #00FFh,A,A   ; Mask to low byte only (actually hardware
                          ; only uses bits 7:0 of data bus, should not
                          ; need to mask)
    stlm     A,AR3
    portw    AR3,wr_rx+lar  ; Write to low address register

;     ld       #0020h,A   ; Space over
;     call     asx

    ld    *AR2+,A   ; Get next table word (dr4 and dr3 bytes)
    ld    A,B       ; Save a copy

;     call     dis4hex        ; Display

    sftl     A,-8,A   ; Shift high byte to low byte
    stlm     A,AR3    ; AR3 holds output data
    portw    AR3,wr_rx+dr4  ; Store to AD6620 MS data byte register

    ld       B,A          ; Get copy
    and      #00FFh,A,A   ; Mask to low byte only
    stlm     A,AR3
    portw    AR3,wr_rx+dr3

;     ld       #0020h,A   ; Space over
;     call     asx

    ld    *AR2+,A   ; Get next table word (dr2 and dr1 bytes)
    ld    A,B       ; Save a copy

;     call     dis4hex  ; Display

    sftl     A,-8,A   ; Shift high byte to low byte
    stlm     A,AR3
    portw    AR3,wr_rx+dr2

    ld       B,A          ; Get copy
    and      #00FFh,A,A   ; Mask to low byte only
    stlm     A,AR3
```

```
    portw    AR3,wr_rx+dr1

;    ld      #0020h,A    ; Space over
;    call    asx

    ld    *AR2+,A      ; Get next table word (dr0 in upper byte)

;    call    dis4hex   ; Display

    sftl    A,-8,A  ; Shift high byte to low byte
    stlm    A, AR3  ; Save for subsequent output port write
    portw   AR3,wr_rx+dr0  ; Address for RSP LS data byte

;    ld    #0020h,A    ; Space over
;    call    asx
;    ld      #000Dh,A  ; Output CR
;    call    asx
;    ld      #000Ah,A  ; LF
;    call    asx

    b    rsp_loop  ; Go back for next table entry
rspx:
;    ld      #ad6620_msg2,A
;    call    msgout
    ret
```

A.3.3 Data Processing Functions

Below lies a subset of the data processing functions used by the AGO code. These are the codes written by in-house, not ones provided in or adapted from the TI DSP library.

`scale.asm` This file provides three functions: two stages of post-FFT scaling, first before the square-magnitude function, and then again before the logarithm, and then the descaling function for post-logarithm.

```
;```````
; Prescaling functions by Micah P. Dombrowski
;
; _sqmag_prescale
;
; Used on 32-bit complex number array (stored RIRIRI), finds the
; largest possible shift applicable to each RI pair using EXP.
; Assumes a zero return equates to EXP(0), and stores the maximum
; shift.  Stores 2*shift in scale factor array.
;
; _log_prescale
```

```
;
; For TI DSP Library Logarithm: normalizes each 32-bit value using
; EXP and NORM, adding shift values to existing values in the save
; array, and cutting to 16-bit output.
;
; _descale
;
; Adjusts logarithmic output based on scale factor array, by
; subtracting scale*log10(2).
;_______

    .mmregs

; Stack usage
; 0 = ST1, 1 = ST0, 2 = function return pointer
    .asg    *SP(3), idata
    .asg    *SP(4), odata
    .asg    *SP(5), sdata

    .def    _sqmag_prescale, _log_prescale, _descale
    .sect   .scale_p

;```````
; _sqmag_prescale
;
; Inputs:       N, number of values to scale in Acc,
; Top of Stack: data input address (512x2 words present),
;               data output address (512 words free),
;               scale save address (512 words free)

_sqmag_prescale

; Set up processor for signed, non-fractional math
    pshm    ST0
    pshm    ST1
    ssbx    CPL
    rsbx    FRCT
    ssbx    SXM
    ssbx    OVM
    rsbx    C16
    nop
    nop

    sub     #1, A   ; BRC = N-1
    stlm    A, BRC
    stm     #16, AR0     ; max shift value
    mvdk    idata, AR2   ; input pointer
```

```
    mvdk    odata, AR3  ; output pointer
    mvdk    sdata, AR4  ; scale array pointer

    rptb    sqmag_prescale_loop - 1

    dld     *AR2+, A
    dld     *AR2-, B
    nop
    nop

    exp     A
    nop
    ldm     T, A

    exp     B
    nop
    ldm     T, B

    min     A       ; A = min(A,B)
    nop
    nop

    sub     #4, A   ; 4 guard bits


    stlm    A, T   ; re-store to T
    nop
    nop

    pshm    T   ; save T to stack

    dld     *AR2+, A
    dld     *AR2+, B
    nop
    nop

    norm    A   ; shift
    norm    B

    .global norm_ovm
norm_ovm:

    dst     A, *AR3+   ; save data
    dst     B, *AR3+

    ld      #0, A   ; clear Acc
    popm    AL      ; pop the corrected scale factor into low Acc
```

~~~
    stl     A, 1, *AR4+  ; save with a 1-bit shift (mpy by 2)

sqmag_prescale_loop:

    popm    ST1
    popm    ST0

    retd
    nop
    nop


;```````
; _log_prescale
;
; Inputs:        N, number of values to scale in Acc,
; Top of Stack:  data input address (512x2 words present),
;                data output address (512 words free),
;                scale save address (512 words free)

_log_prescale

; Set up processor for fractional, signed math
    pshm    ST0
    pshm    ST1
    ssbx    CPL
    ssbx    FRCT
    ssbx    SXM
    ssbx    OVM
    rsbx    C16
    nop
    nop

    sub     #1, A       ; BRC = N-1
    stlm    A, BRC
    stm     #16, AR0    ; max shift value
    mvdk    idata, AR2  ; input pointer
    mvdk    odata, AR3  ; output pointer
    mvdk    sdata, AR4  ; scale array pointer

    rptb    log_prescale_loop - 1

    dld     *AR2+, A
    exp     A
    nop

    ldm     T, B      ; load T
~~~

~~~
    sub     #4, B     ; guard bits
    stlm    B, T
    add     *AR4, B   ; add any existing scale factor
    stl     B, *AR4+  ; save back to scale array
    nop
    nop

    norm    A  ; shift

    sth     A, *AR3+  ; save data

log_prescale_loop:

    popm    ST1
    popm    ST0

    retd
    nop
    nop


;```````
; _descale
;
; Inputs:       N, number of input points, in Acc
; Top of Stack: data input address (512x2 words present, Q16.15 format)
;               data output address    (512 words free, in-place okay)
;               scale factor array (512 words present)

log10o32767    .set    0x783F  ; ( log10(32767) * 2^15) >> 3
log10o2        .set    0x04D1  ; ( log10(2) * 2^15 ) >> 3


_descale:

    pshm    ST0
    pshm    ST1
    ssbx    CPL
    rsbx    FRCT
    ssbx    SXM
    rsbx    OVM
    rsbx    C16
    nop
    nop

    sub    #1, A    ; BRC = N-1
    stlm    A, BRC
~~~

~~~
    mvdk    idata, AR2    ; input pointer
    mvdk    odata, AR3    ; output pointer
    mvdk    sdata, AR4    ; scale array pointer

    rptb    descale_loop - 1

    ; Docs say log10 outputs Q16.15, but this is misleading,
    ; format is S IIII IIII IIII IIII FFFF FFFF FFFF FFF
    dld     *AR2+, A
    sfta    A, #-3

    ld      #log10o32767, B
    add     B, A

    ld      #log10o2, B

    rpt     *AR4+
    sub     B, A
    sfta    A, #8
    sfta    A, #8
    sat     A

    sth     A, *AR3+

descale_loop:

    popm    ST1
    popm    ST0

    retd
    nop
    nop


    .end
~~~

`dpsm.asm` This provides a single function, $|C^2|$.

~~~
;```````
; Double-precision square magnitude function by Micah P. Dombrowski
;
; Reads n Q.31 numbers arrayed as R[0], I[0], R[1], I[1], ..., R[n-1],
; I[n-1] outputs MSB half of R[0]^2+I[0]^2, R[1]^2+I[1]^2, ...,
; R[n-1]^2+I[n-1]^2 output fills first half of input region.
;
; Inputs: data address in A, number of R/I pairs in B

~~~

```
    .mmregs
    .def    _sqmag
    .sect   .dpsm_p
_sqmag

    pshm    ST0
    pshm    ST1
    ssbx    SXM
    ssbx    FRCT
    ssbx    OVM
    rsbx    C16
    nop
    nop

; Double-precision square magnitude, saving MSB half of result.

    stm     #0, T    ; Multiplication Temp register (for mpy)
    stm     #0, BK   ; Circuluar addressing modulus (do not want)
    sub     #1, B
    stlm    B, BRC
    stm     #2, AR0  ; Increment (jump to next 32-bit datum)
    stlm    A, AR2   ; Load index
    stlm    A, AR3   ; Load index
    stlm    A, AR4   ; Storage index
    rptb    sqmag_loop - 1

    mpy     *AR2+, A              ; a = 0       (1)
    macsu   *AR2-, *AR3+, A       ; a = RL*RH   (1)
    macsu   *AR3-, *AR2, A        ; a += RH*RL  (1)
    ld      A, -16, A             ; a >>= 16    (1)
    mac     *AR2+0%, *AR3+0%, A ; a += RH*RH  (1)
    stm     #0, T  ; (2)
    sat     A      ; (1)

    mpy     *AR2+, B              ; b = 0       (1)
    macsu   *AR2-, *AR3+, B       ; b = IL*IH   (1)
    macsu   *AR3-, *AR2, B        ; b += IH*IL  (1)
    ld      B, -16, B             ; b >>= 16    (1)
    mac     *AR2+0%, *AR3+0%, B ; b += IH*IH  (1)
    stm     #0, T  ; (2)
    sat     B      ; (1)


    add     B, A     ; a += b == R^2 + I^2
    sat     A
    dst     A, *AR4+ ; (1)

```

~~~
sqmag_loop:

    popm    ST1
    popm    ST0

    nop
    nop

    retd
    nop
    nop
~~~

`tablemax.asm` The function provided by this file reduces the final data set by taking the max between a provided set of indices within the 512-bin FFT. The bins in the table can be spaced by 1 to have ranges of complete data transferred.

~~~~
;```````
; Frequency selection and averaging function by
; Nathan Utterback and Micah P. Dombrowski
;
; Inputs: start address of data Acc,
;         output address in Bcc

avg_shift_val    .set 3 ; bits to right shift by after summing

    .mmregs
    .def     transfer, transfer_table_sz
    .sect    .transfer_p

    .bss     Delta,1,0,0  ; storage for repeat counter
    .bss     nShift,1,0,0 ; storage for shift value

transfer:

     pshm    ST0
     pshm    ST1

     pshm    AR6

     stlm    A, AR2
     stlm    B, AR3
     stm     transfer_table_start, AR5 ; load the start of the
                                       ; table into memory
     stm     transfer_table_end-1, AR0


transfer_sum_loop:
~~~~

```

    ldm     AR2, A    ; load base address
    add     *AR5+, A  ; add offset from table, inc
    stlm    A, AR4    ; store

    ld      *AR5-, A  ; load next offset, dec
    sub     *AR5+, A  ; subtract current offset to get delta, inc
    sub     #1, A
    stlm    A, BRC    ; store delta-1 for rpt

    ld      #0, B
    rptb    max_loop - 1

    ld      *AR4+, A
    max     B

max_loop:

    stl     B, *AR3+  ; save

    cmpr    LT, AR5
    bc      transfer_sum_loop, TC  ; loop until we finish the table

    popm    AR6

    popm    ST1
    popm    ST0

    retd
    nop
    nop

transfer_table_start:
   .word    25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105

   .word    113, 114, 115, 116, 117, 118, 119, 120, 121
   .word    122, 123, 124, 125, 126, 127, 128, 129, 130
   .word    131, 132, 133, 134, 135, 136, 137, 138, 139
   .word    140, 141, 142, 143, 144, 145, 146, 147, 148
   .word    149, 150, 151, 152, 153, 154, 155, 156, 157
   .word    158, 159, 160, 161, 162, 163, 164, 165

   .word    166, 174, 182, 190, 198, 206, 214, 222, 230
   .word    238, 246, 254, 262, 270, 278, 286, 294, 302
   .word    310, 318, 326, 334, 342, 350, 358, 366, 374
   .word    382, 390, 398, 406, 414, 422, 430, 438, 446
   .word    454, 462, 470, 478
```

```
transfer_table_end:

transfer_table_sz    .set    transfer_table_end-transfer_table_start
```

`sercook.asm` Finally, this provides a function to change the data into the right form for serial output.

```
;```````
; Serial data cooking function by Micah P. Dombrowski
;
; Reads N words containing right-aligned bytes,
; bit reverses, and adds start and stop bits.
;
; Inputs: data address in A, number of bytes in B


    .mmregs
    .def    _serial_cook
    .sect    .sercook_p
_serial_cook

    sub    #1, B
    stlm    B, BRC
    stlm    A, AR0
    rptb    bitrev_loop - 1

    ssbx   XF

    ld     #1, B     ; zero result + stop bit

    ld     #001h, A  ; load mask
    and    *AR0, A   ; mask data
    or     A, 8, B   ; OR into result
    ld     #002h, A
    and    *AR0, A
    or     A, 6, B
    ld     #004h, A
    and    *AR0, A
    or     A, 4, B
    ld     #008h, A
    and    *AR0, A
    or     A, 2, B
    ld     #010h, A
    and    *AR0, A
    or     A, 0, B
    ld     #020h, A
    and    *AR0, A
```

```
    or      A, -2, B
    ld      #040h, A
    and     *AR0, A
    or      A, -4, B
    ld      #080h, A
    and     *AR0, A
    or      A, -6, B

    stl     B, *AR0+    ; rewrite to serial buffer

    rsbx    XF

bitrev_loop:

    retd
    nop
    nop

    .end
```

A.3.4 S-ARC Main Program Code

The main code which ties all of the above—as well as the windowing, FFT, and log10 functions—together.

```
;```````
;
; Dartmouth College AGO Rx-DSP Program
;
; Written by: Micah P. Dombrowski and Nathan B. Utterback
;     w/ code segments from TI DSP Library
;
;_______

    .mmregs
    .global ZERO, BMAR, PREG, DBMR, INDX, ARCR, TREG1
    .global TREG2, CBSR1, CBER1, CBSR2, CBER2
    .global RXDSP_START
    .ref    _cbrev32, _cfft32_512, _log_10
    .ref    _hann_window, _sqmag, _serial_cook
    .ref    _log_prescale, _sqmag_prescale, _descale
    .ref    rsp_clear, rsp_reset, rsp_init, rsp_mstart, rsp_sstart
    .ref    transfer, transfer_table_sz
    .global bridge_data, buff_clear_loop
    .def    ago_main, int_nmi

```

```
    .include "rx-dsp.h"
    .text

code_version  .string  "v1.0"
band_width    .string  "0300"

; Output constants

output_shift_n    .set  8       ; left shift before 8-bit mask defines
header_freq_mask  .set  0x0FFF  ; bits of 32-bit major frame counter
                                ; that must be zero for a header frame

; Run constants
fft_scaling    .set  0
data_n         .set  512  ; Size of each FFT (# of IQ pairs)
data_discard   .set  512  ; words discarded from Rx FIFO pre-data
data_minor_sz  .set  1    ; acquisitions per half-buffer interrupt
fsync_sz       .set  4    ; # of serial frame sync bytes
                          ; (should be multiple of 4)
abu_buff_sz    .set  214  ; size of serial buffer
                          ; (2x major frame size IN BYTES)
                          ; should be set to
                          ; 2*(transfer_table_sz + fsync_sz)

; Memory allocations
data_addr      .usect  ".data_v", 0x800, 1, 1
scale_addr     .usect  ".scale_v", 0x200, 1, 1
stackres       .usect  ".stack_v", 0x40, 1, 1
abu_buff_loc   .usect  ".sbuff_v", abu_buff_sz, 1, 1
abu_buff_hloc  .set    abu_buff_loc+abu_buff_sz/2  ; half-way

; Memory pointers
iq_data     .set    data_addr     ; 512 * 2 words * I/Q
fft_data    .set    data_addr     ; 512 * 2 words * Re/Im
scale_data  .set    scale_addr    ; 512 words
sqmag_data  .set    data_addr     ; 512 * 2 words
sqsc_data   .set    data_addr+2*data_n    ; 512 words
log_data    .set    data_addr     ; 512 * 2 words
power_data  .set    data_addr     ; 512 words
ebs_data    .set    data_addr+data_n      ; 512 words

; mode flags
mode_std_bit    .set    0001b    ; standard operations
mode_dbg_bit    .set    0010b    ; debug

mode_std_n        .set    transfer_table_sz
mode_dbg_n        .set    512
```

```

; Scratchpad RAM usage
bridge_count    .set    scratch
minor_count     .set    scratch+1
bspce_save      .set    scratch+2
bridge_size     .set    scratch+3
mode_flag       .set    scratch+4
shb_addr        .set    scratch+5
major_count     .set    scratch+6    ; two words!
nco_freq        .set    scratch+8    ; two words!

    .bss TempLmem,1*2,0,0  ;temporary dword


RXDSP_START
ago_main:

    rsbx    XF

; Processor setup
    ssbx    INTM          ; Disable interrupts
    stm     #(stackres+0x40), SP  ; set Stack Pointer
    stm     #npmst,PMST   ; Set processor mode/status
;    stm    #defst0, ST0
;    stm    #defst1, ST1
    rsbx    SXM           ; Suppress sign extension
;    rsbx   XF
    nop     ; Space for branch to app
    nop

appcode:
;    stm    #0,state   ; Clear interrupt routine state
    stm    #0,AR0     ; Clear auxilliary register 0

    portw    AR0,wr_disc  ; Enable parallel TLM drivers, I_Q out
    portw    AR0,wr_dog   ; Strobe watchdog timer

    stm    #0,AR0  ; Clear all auxiliary registers
    stm    #0,AR1
    stm    #0,AR2
    stm    #0,AR3
    stm    #0,AR4
    stm    #0,AR5
    stm    #0,AR6
    stm    #0,AR7

    stm    #0FFh,IFR  ; Clear any pending interrupts
```

```
    stm    #ntss,TCR  ; Stop timer, if running

; Main data code start
read_init:
    call    rsp_reset
    nop
    nop
    portw    AR0,wr_rs_rx  ; Hardware reset of AD6620 RSP
    portw    AR0,wr_rs_rx  ; Hardware reset of AD6620 RSP
    portw    AR0,wr_rs_rx  ; Hardware reset of AD6620 RSP
    portw    AR0,wr_rs_rx  ; Hardware reset of AD6620 RSP
    portw    AR0,wr_rs_rx  ; Hardware reset of AD6620 RSP
    portw    AR0,wr_rs_rx  ; Hardware reset of AD6620 RSP
    nop
    nop
    call    rsp_reset
    call    rsp_init
    call    rsp_clear
    call    rsp_mstart

    rpt    #4444  ; Let the AD6620 do its first initialization in peace
    nop

    call    rsp_reset

; store permanent NCO Frequency
    ld     #0x4420, A
    stl    A, @nco_freq
    ld     #0x01FA, A
    stl    A, @(nco_freq+1)

    portr    rs_rx_fifo, AR0   ; Reset RxFIFO - also wired to Slave
    nop
    nop

    stm      #lsb_sel, AR0  ; Reset acq_seq
    portw    AR0, wr_disc
    nop

    ; BSP prep

    stm    #(bspc_Free+bspc_fsm), BSPC0   ; reset BSP
    stm    #(int_bx), IMR                 ; unmask serial tx interrupt
    stm    #(bspce_fe+bspce_bxe), BSPCE0  ; 10-bit words,
                                          ; enable tx autobuffer

    ; where in the 2 kw of buffer RAM does the transmit buffer start?
```

```
    stm    #(abu_buff_loc-0x800), AXR
    stm    #(abu_buff_sz), BKX  ; buffer size

; Clear entire serial buffer
    stm    #abu_buff_sz-1, BRC
    stm    #abu_buff_loc, AR4
    rptb   buff_init_loop - 1

    st     #0h, *AR4+

buff_init_loop:
    .global buff_init_loop, head_ramp, major_loop

; Write out header to first buffer half

    stm    #abu_buff_loc, AR4
    ; two-byte frame sync 0xEB90
;    stm    #(abu_buff_loc+abu_buff_sz/2-2), AR4
    st    #0xFE, *AR4+  ; 4-byte initialization sync
    st    #0x6B, *AR4+
    st    #0x28, *AR4+
    st    #0x40, *AR4+

    stm    #file_header, AR0  ; Point to static header words
header_loop:
    ld     *AR0+, A            ; Get a word, point to next
    bc     header_loop_x, AEQ  ; If terminator, end static header
    stl    A, *AR4+            ; Write to serial buffer
    b      header_loop
header_loop_x:

    ld    #abu_buff_loc, A
    ld    #(abu_buff_sz/2), B

    call    _serial_cook

; Set initial bspce_save such that the first acquisition will
; write to the second half of the serial buffer
    st    #bspce_xh, @bspce_save

; Start BSP transmits
    ; have to hold fsm bit
    stm    #(bspc_Free + bspc_fsm + bspc_nXrst), BSPC0

;    rsbx    INTM  ; global interrupt enable

; All Aux Registers are fungible in the main loop: values which must
```

```
; be preserved over time are stored in the scratchpad RAM as defined
; above.  Note that only AR6 and AR7 are required to be preserved by
; the DSP Library functions (and most others), all other ARx may be
; modified within function calls.

    ld      #0, A
    dst     A, @major_count

major_loop:

    portr    rs_rx_fifo, AR0  ; Reset Rx FIFO - also wired to Slave
    nop
    nop

    stm      #(acq_seq_out+lsb_sel), AR0  ; send acq_seq, set lsb_sel
    portw    AR0, wr_disc

    call    rsp_clear    ; clear NCO RAM
    call    rsp_mstart    ; and start digitizing

;
; Reads
;
data_acq_start:

    st    #0, @bridge_count  ; reset bridged data counter
    st    #0, @minor_count   ; set minor frame counter

;
; Set wait states for Rx FIFO
;
    ldm   SWWSR, A
    ld    #65535, B
    xor   #7, #swwsr_is, B ; (0b111<<Nset XOR 0d65535) creates bitmask
    and   B, A             ; mask out bits of interest
    or    #0, #swwsr_is, A ; (A or Nwait<<Nset) to set Nwait to Nset
    ; Nwait = 0 means no additional waits
    stlm    A, SWWSR

    ssbx    XF
    rpt     #100
    nop
    rsbx    XF

    .global    pre_disc, pre_read
pre_disc:

```

```
; loop to read and discard first data out of AD6620
    stm    #(data_discard), AR2
    nop
rx_discard_loop:
    ; read only if rx fifo is nonempty
    portr    rd_disc, AR0
    nop
    nop
    bitf    AR0, #rx_efo

    bc    discard_fifo_empty, NTC
    portr    rd_rx_out, AR1   ; read data into AR1
    mar    *AR2-              ; decrement word counter
discard_fifo_empty:

    banz    rx_discard_loop, *AR2


; loop to read data from Rx FIFO into RAM
    stm    #2, AR0
    stm    #iq_data, AR1   ; set address for first data word
    stm    #(data_n*2-1), AR2
pre_read:
    nop
rx_read_loop:
    ; read only if rx fifo is nonempty
    portr    rd_disc, AR3
    nop
    nop
    bitf     AR3, #rx_efo

    bc    read_fifo_empty, NTC

    portr    rd_rx_out, *AR1+
    st     #0, *AR1+  ; zero out second word
    mar    *AR2-      ; decrement word counter

read_fifo_empty:

    banzd    rx_read_loop, *AR2
    nop
    nop

;
; Set wait states for other stuff (full 7)
;
    ldm    SWWSR, A
```

```
    or      #7, #swwsr_is, A   ; (A or Nwait<<Nset) to set Nwait to Nset
    stlm    A, SWWSR
    nop
    nop

    ssbx     XF
    rpt      #50
    nop
    rsbx     XF

;
; End data acquisition, begin data processing
;

data_process:


    .global pre_window
pre_window:

    ld     #iq_data, A
    ld     #data_n, B

    call     _hann_window

    .global     pre_bit_rev
pre_bit_rev:

; Bit reversal
    stm      #data_n, AR0
    pshm     AR0
    stm      #iq_data, AR0
    pshm     AR0
    ld       #iq_data, A

    call     _cbrev32

    frame    2


    .global pre_fft
pre_fft:
    stm      #fft_scaling, AR0
    pshm     AR0
    ld       #fft_data, A

    call     _cfft32_512
```

```

    frame   1


    .global    pre_move
pre_move:
; flip things into proper power spectra order (swap halves)
; remove zeroes here, too

    stm     #(data_n-1), BRC
    stm     #fft_data, AR2
    stm     #(fft_data + 2*data_n), AR3

    rptb    move_loop - 1

    dld     *AR2, A
    dld     *AR3, B

    nop
    nop
    xc      2, AEQ
    add     #1, A
    xc      2, BEQ
    add     #1, B

    dst     A, *AR3+
    dst     B, *AR2+

move_loop:

    .global    pre_sqscale
pre_sqscale:

    stm     #scale_addr, AR0  ; scale saves
    pshm    AR0
    stm     #fft_data, AR0    ; output
    pshm    AR0
    stm     #fft_data, AR0    ; input
    pshm    AR0
    ld      #data_n, A  ; N

    call    _sqmag_prescale

    frame   3  ; free stack


    .global pre_abs
```

```
pre_abs:
    ssbx    SXM
    ssbx    OVM
    nop
    nop

    stm     #(2*data_n-1), BRC
    stm     #fft_data, AR0
    rptb    abs_loop - 1

    dld     *AR0, A
    abs     A
    dst     A, *AR0+

abs_loop:

    .global    pre_sqmag, pre_log, pre_db
pre_sqmag:
; |FFT|^2
    ld    #sqmag_data, A
    ld    #data_n, B

    call    _sqmag

    .global pre_logps
pre_logps:

    stm     #scale_data, AR0  ; scale saves
    pshm    AR0
    stm     #sqsc_data, AR0   ; output
    pshm    AR0
    stm     #sqmag_data, AR0  ; input
    pshm    AR0
    ld      #data_n, A  ; N

    call    _log_prescale

    frame   3


pre_scale:


pre_log:
; log_10(|FFT|^2) (outputs 32-bit Q16.15)

    stm     #data_n, AR0
```

```
    pshm    AR0
    stm     #log_data, AR0  ; write to beginning of data buffer
    pshm    AR0
    ld      #sqsc_data, A   ; read from halfway point of data buffer

    call _log_10

    frame    2


    .global pre_descale
pre_descale:

    stm     #scale_data, AR0  ; scale saves
    pshm    AR0
    stm     #power_data, AR0  ; output
    pshm    AR0
    stm     #log_data, AR0    ; input
    pshm    AR0
    ld      #data_n, A  ; N

    call    _descale

    frame   3  ; free stack


    .global post_descale
post_descale:
; multiply by output_scale_factor,
; shift right by output_shift_n, re-store

    pshm    ST0
    pshm    ST1
    ssbx    FRCT
    ssbx    SXM
    ssbx    OVM
    rsbx    C16

; Scale and shift, save 8-bit data

    stm     #data_n-1, BRC
    stm     #power_data, AR0
    stm     #(data_addr + 2*data_n), AR1
    stm     #ebs_data, AR2
;    stm     #output_scale_factor, T
    rptb    ebs_loop - 1

```

```
;     mpy     *AR0+, A   ; multiply by scale factor in T
;     sfta    A, #0-output_shift_n
;     and     #0xFF0000, A
;     dst     A, *AR1+

;     mpy      *AR0+, A   ; multiply by scale factor in T
    ld      *AR0+, A
    sfta    A, #0-output_shift_n
    add     #128, A
    and     #0xFF, A
    stl     A, *AR2+
;     dadd     output_shift_n, A   ; shift
;     sat      A
;     and      #0xFF, #16, A   ; mask to A(23-16)
;     sth      A, *AR2+        ; store A(23-16)

ebs_loop:

    nop

    .global    dp_end
dp_end:

; Debug code, writes a 512-byte ramp-up-ramp-down
;     stm      #data_n/2-1, BRC
;     stm      #0, AR0
;     stm      #ebs_data, AR2
;     stm      #ebs_data+data_n-1, AR3
;     rptb     dummy_data - 1
;
;     mvkd     AR0, *AR2+
;     mvkd     AR0, *AR3-
;     mar         *AR0+
;
;dummy_data:
;     .global dummy_data

    popm    ST1
    popm    ST0

    nop

;
; End data processing, begin serial data handling
;

    stm      #(lsb_sel), AR0
```

```
    portw    AR0, wr_disc

; Standard mode (#buff_size bytes) or
; debug mode (#debug_size) depending on trm_28 state.
    portr    rd_disc, AR0    ; Get discrete bits

;    stm    #0, AR0    ; DEBUG !!
;    nop
;    nop

    bitf    AR0, #trm_28  ; Test for high terminal input
    bc      standard_mode, NTC  ; If trm_28 is low (NTC), standard
                                ; data settings to #debug_size

    st    #mode_dbg_n, @bridge_size
    st    #mode_dbg_bit, @mode_flag

    b     debug_mode_skip

standard_mode:

    st    #mode_std_n, @bridge_size
    st    #mode_std_bit, @mode_flag

debug_mode_skip:

; entry point for bridging data transfers
; over multiple serial half-buffers
bridge_data:
    stm      #(acq_test2+lsb_sel), AR0  ; send acq_seq, set lsb_sel
    portw    AR0, wr_disc

    nop
    nop
    ; Determine serial buffer position
    ld    #abu_buff_loc, A  ; load buffer base

    bitf    @bspce_save, #bspce_xh  ; read XH out of
                                    ; stored BSPCE register
    nop
    nop
    bc      buff_skip, NTC  ; if first half _finished_
                            ; (XH=0, NTC), do nothing

    add      #(abu_buff_sz/2), A
    stm      #(acq_test2+acq_test3+lsb_sel), AR0
    portw    AR0, wr_disc
```

```

buff_skip:
    nop
    nop
    stl  A, @shb_addr ; scratch storage for serial half-buffer address
    nop
    nop

abu_first_half:
    .global abu_first_half

    ssbx    INTM

; Clear serial buffer half
    mvdm    @shb_addr, AR4
    nop
    rpt     #(abu_buff_sz/2-1)
    st      #0xFF, *AR4+

; reset AR4 for data copy
    mvdm    @shb_addr, AR4
    nop
    nop

abu_fill_start:
    .global abu_fill_start

    ; two-byte frame sync 0xEB90
    st    #0xEB, *AR4+
    st    #0x90, *AR4+

    ; two-byte infofoop
    ld     @minor_count, A   ; byte 1, minor frame number
    and    #0xFF, A
    stl    A, *AR4+
    dld    @major_count, A   ; byte 2, major frame number
    and    #0xFF, A, B
    stl    B, *AR4+

post_sync_write:
    .global post_sync_write

; Transfer in selected data mode
    bitf    @mode_flag, #mode_dbg_bit
    bc      dbg_transfer, TC

; In standard mode, check if 8-bit major_count (still in A) == 0,
```

```
; if so, transfer a header instead of data.

; Std header: spit out a header frame every 4096th
    and    #header_freq_mask, A
    bc     header_skip, ANEQ  ; A != 0, skip header

    call    hwrite
    st      #mode_std_n, @bridge_count  ; fake it out
    nop
    nop

    b    end_transfer

header_skip:
; Std transfer: selected bins in a 1-frame major frame
    ; transfer selected data to serial buffer
    ld     #ebs_data, A  ; input addr in A
    ldm    AR4, B        ; output addr in B

    call   transfer

    st    #mode_std_n, @bridge_count  ; fake it out

    b     end_transfer

; Debug transfer: entire 512-bin fft
; spread over multiple minor frames.
dbg_transfer:

    ; copy raw data (words) into serial buffer (bytes)
    ld      #ebs_data, A
    add     @bridge_count, A
    stlm    A, AR2

    mvdm    @bridge_size, AR0    ; goal bridge size
    mvdm    @bridge_count, AR1    ; current count

    stm     #((abu_buff_sz/2-fsync_sz)-1), BRC
    rptb    rawdata_loop - 1

;    ld      *AR2+, A   ; load (data word) to Acc
;    and     #0xFF, A   ; mask to low-byte
;    stl     A, *AR4+   ; save to serial buffer

    mvdd    *AR2+, *AR4+

    mar     *AR1+
```

```
    cmpr    LT, AR1  ; If we're not done with a bridged data sequence,
    nop
    nop
    xc      2, NTC
    rsbx    BRAF
    nop
    nop
    nop
    nop
    nop
    nop
    .global rawdata_loop, dbg_transfer_skip
rawdata_loop:

    mvmd    AR1, @bridge_count
    nop

end_transfer:
    .global    end_transfer

    nop

serial_transfer_end:
    .global    serial_transfer_end
    nop

; Bit reverse and add start/stop bits
    ld    @shb_addr, A
    ld    #(abu_buff_sz/2), B

    call    _serial_cook

;    rsbx    INTM

    addm    #1, @minor_count

; If a major frame is complete, shut it down

; unset acq_seq, keep lsb_sel
    stm      #lsb_sel, AR0
    portw    AR0, wr_disc

; Strobe watchdog- once per acquisition
    stm      #0, AR0     ; Data is not used- just the wr_dog strobe
    portw    AR0,wr_dog  ; Strobe the watchdog

; Stop acquisition, clear interrupts, then idle until an interrupt.
```

```
    call    rsp_reset

    nop
    nop

    .global pre_sleep
pre_sleep:

    stm      #(acq_test4+lsb_sel), AR0
    portw    AR0, wr_disc

    ssbx    INTM
    stm     #0FFh,IFR  ; Clear any pending interrupts

    idle    2  ; and now...we wait.

    .global post_sleep
post_sleep:

; check for aux int -> serial monitor
;    bitf     IFR, #int_3
;    cc       inth_3, TC

; make sure we had a serial interrupt
    bitf    IFR, #int_bx
    bc      pre_sleep, NTC  ; stray interrupt, go back to IDLE

    nop

    stm    #int_bx, IFR  ; clear int flag

    mvmd  BSPCE0, @bspce_save ; store control extension register in AR6

    bitf    BSPCE0, #bspce_xh
    bc      xh_skip, NTC

    stm      #(lsb_sel), AR0
    portw    AR0, wr_disc

xh_skip:

    rpt    #100
    nop

    stm      #lsb_sel, AR0
    portw    AR0, wr_disc

```

```
    mvdm    @bridge_size, AR0   ; need to copy these
    mvdm    @bridge_count, AR1  ; to use CMPR
    nop
    nop
    cmpr    LT, AR1  ; If we're not done with a bridged data sequence,
    bc      bridge_data, TC  ; jump to bridge_data to
                             ; continue transfers, otherwise...
    dld     @major_count, A  ; increment major frame counter
    add     #1, A
    dst     A, @major_count

    b    major_loop  ; new data acquisition

;
; Main acquisiton ('appcode') branch done
;

;
; Interrupts
;

; Non-Maskable Interrupt
;     this is hit by the watchdog
int_nmi:
    nop
    nop

    stm      #0, AR0
    portw    AR0,wr_dog  ; Strobe watchdog timer

    b    0xF800

; Setup: IPTR=0x1FF, OVLY=1, all else =0
; This should set things up to completely
; reload the program from the EPROM on reset.
    stm    #0xFFA0, PMST
    nop
    nop

;    reset  ; I don't have to take this.  ...I'm going home.

    ret  ; should never get here!

inth_3:
    ssbx    INTM
    stm     #int_3, IFR  ; clear int flag

```

```
    ; call serial monitor

;    calld   _RxDSP_Monitor
;    stm     #bspce_haltx, BSPCE   ; tell serial transmit to halt
                                   ; after completing this half-buffer

    retd
    nop
    nop

file_header:
    .string    "Dartmouth College Rx-DSP, AGO Site 3 Unit 0."
    .word      0000h   ; Null terminator

;```````
; hwrite
;
; Writes a header:
;
; <0xFE6B2840><RxDSP><Unit #><Ver #><NCOF><MFCB><00000000>
;_______

hwrite:
    .global    hwrite

    pshm       AR3

; 4-byte sync
    stm     #static_header, AR3   ; Point to static header words
    rpt     #static_header_len    ; <SYNC><RxDSP>
    mvdd    *AR3+, *AR4+

    stm     #code_version, AR3
    rpt     #3
    mvdd    *AR3+, *AR4+

    stm     #spec_header, AR3   ; Point to static header words
    rpt     #spec_header_len    ; <skipNS><fbstFBSN><fbenFBEN>
    mvdd    *AR3+, *AR4+

    stm     nco_freq, AR3
    ld      *AR3, #-8, A
    and     #0xFF, A
    stl     A, *AR4+
    ld      *AR3+, A   ; inc to second word
    and     #0xFF, A
    stl     A, *AR4+
```

```
    ld      *AR3, #-8, A
    and     #0xFF, A
    stl     A, *AR4+
    ld      *AR3, A
    and     #0xFF, A
    stl     A, *AR4+

    stm     #band_width, AR3
    rpt     #3
    mvdd    *AR3+, *AR4+

    stm     major_count, AR3
    ld      *AR3, #-8, A
    and     #0xFF, A
    stl     A, *AR4+
    ld      *AR3+, A   ; inc to second word
    and     #0xFF, A
    stl     A, *AR4+
    ld      *AR3, #-8, A
    and     #0xFF, A
    stl     A, *AR4+
    ld      *AR3, A
    and     #0xFF, A
    stl     A, *AR4+

    popm    AR3

    retd
    nop
    nop

static_header:
    .word      0xFE, 0x6B, 0x28, 0x40
    .string    "AGORxDSP"  ; 12 bytes
static_header_end:
static_header_len    .set    static_header_end-static_header-1

spec_header:
    .string    "stride08"
    .string    "cbst0113"
    .string    "cben0166"
    .string    "00000000"  ; pad to 32 bytes
spec_header_end:
spec_header_len    .set    spec_header_end-spec_header-1

    .end
```

File Frame \Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	…	N-1	N
1	FS1	FS2	FS3	FS4	FS5	FS6	FC1	FC2	TSs1	TSs2	TSs3	TSs4	TSs5	TSu1	TSu2	TSu3	AD1	AD2	…	ADn-1	ADn
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

	128 Bits								
File Frame Header Size	16 Bytes								
	10	11	12	13	14	15	16	17	
File Frame Data Size	1	2	4	8	16	32	64	128	KB
Percent Overhead	1.56	0.78	0.39	0.2	0.1	0.05	0.02	0.01	%

FS: 48-bit File Sync bit pattern = 0xb0f919c5025d
FC: 16-bit File Frame Counter
TSs: 40-bit seconds since epoch
TSu: 24-bit microseconds
ADn: serial data bytes

Header Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Data	\xFE	\x6B	\x28	\x40	R	x	D	S	P	Un	S	S	v	1	.	0

Header Byte #	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Data	N	C	O	F	B	A	N	D	M	F	C	B	0	0	0	0

\xFE\x6B\x28\x40RxDSPUnSSv1.0
Un: Unit Number in ASCII

NCOFBANDMFCB0000
NCOF: Center Frequency | BAND: Width of the filter band
MFCB: Major Frame Counter Bytes

Major Frame

Minor Frame # \Byte #	1	2	3	4	5	6	7	8	…	517	518	519	520	521	522	523	524
1	S1	S2	C1	C2	H1	H2	H3	H4	…	I121H	I121L	Q121H	Q121L	I122H	I122L	Q122H	Q122L
2	S1	S2	C1	C2	I123H	I123L	Q123H	Q123L	…		Q66H	Q66L		I128H	I128L	Q128H	Q128L
3	S1	S2	C1	C2	…	…	…	…	…	…	…	…	…	…	…	…	…
4	S1	S2	C1	C2	I449H	I449L	Q449H	Q449L	…	I450L	Q450H	Q450L		I512H	I512L	Q512H	Q512L

Major Frame has 32-byte header, followed by 2048 bytes of data.
Minor Frames are determined by 54x BSP half-buffer size, for above major frame size we use 524-byte minor frames.
Minor frames have 2-byte counters, data, then 2-byte sync words.
C1: Composite info byte, 2 MSB unit designation, 6 LSB minor frame count
C2: Second info byte, major frame count
S1: Sync 1, 0xEB
S2: Sync 2, 0x28
H1…H32: major frame header
(Q/I)1(H/L)…512: Data

Figure A.1: Maps of the Sondrestrom MI-ARC data structure, with file structure top, headers middle, and major/minor frame structure bottom.

A.4 Sondrestrom MI-ARC

The Sondrestrom MI-ARC has three sample-synchronized Rx-DSPs. It cycles through a table of center frequencies, returning small frames of data through a single serial line. The data from the three units is combined on the line by a custom hardware multiplexer. The two unique parts of this deployment's code are an additional RSP function, and the main program code.

A.4.1 Data Structure

Figure A.1 shows the structure of the Sondrestrom data. The data file structure is on top, and file frames include sync words, counters, time data, and the serial data stream (which may be asynchronous to the file's frame structure). Middle is the structure of the header, containing the major frame sync words, unit number, and frequency data. Finally, on the bottom is the internal structure of the major frames, each composed of four minor frames.

A.4.2 RSP One-Shot

This function modifies one memory location in the AD6620's RAM. The most common use for this is to change the center frequency.

~~~~
;```````
; rsp_os    Receive Processor One-Shot
;            Quickly loads one 36-bit value to an address on the AD6620

rsp_os:
    ; 36-bit value in A
    ; 10-bit address in B

    ; write address 0x0303 to address registers
    stl     B, #-8, AR4
    andm    #0x03, AR4  ; mask to lower two bytes
    nop
    nop
    nop
    nop
    portw    AR4, wr_rx+amr

    stl     B, AR4
    andm    #0xFF, AR4
    nop
    nop
    nop
    nop
    portw   AR4, wr_rx+lar

    ; write 36-bit value into five 8-bit data registers
    mvdm    @AG, AR4
    andm    #0x0F, AR4  ; 0x0F 0000 0000
    nop
    nop
    nop
    nop
    portw   AR4, wr_rx+dr4

    sth     A, #-8, AR4  ; 0x00 FF00 0000
    andm    #0xFF, AR4
    nop
    nop
    nop
    nop
    portw   AR4, wr_rx+dr3

    sth     A, AR4  ; 0x00 00FF 0000
~~~~

```
    andm    #0xFF, AR4
    nop
    nop
    nop
    nop
    portw   AR4, wr_rx+dr2

    stl     A, #-8, AR4  ; 0x00 0000 FF00
    andm    #0xFF, AR4
    nop
    nop
    nop
    nop
    portw   AR4, wr_rx+dr1

    stl     A, AR4  ; 0x00 0000 00FF
    andm    #0xFF, AR4
    nop
    nop
    nop
    nop
    portw   AR4, wr_rx+dr0  ; writing to dr0 commits

    retd
    nop
    nop
```

A.4.3 MI-ARC Main Program Code

This main program code is unique in that it was designed to do away with separate codes and `.hex` files for the Master and Slave units. Instead, the code's role is entirely determined by the `unit_designation` uword, which can be targeted by the PROM burner, and auto-incremented when PROMs for a number of units are burned in-sequence.

The code contains two helper functions, `cfreq_walk` and `sync_units`, which implement the frequency switching and frame synchronization, respectively. This setup uses the TCLKR and TCLKX lines of the TDM Serial Port (TSP) as feedback inputs from the two Slave units, to help ensure synchronization.

```
; v1.1    2012.06.01    Added 1-frequency debug mode
;                       controlled by trm_28 jumper

    .mmregs
    .global ZERO, BMAR, PREG, DBMR, INDX, ARCR, TREG1
    .global TREG2, CBSR1, CBER1, CBSR2, CBER2
    .global RXDSP_START
```

```
    .ref    _serial_cook
    .ref    rsp_clear, rsp_reset, rsp_init
    .ref    rsp_mstart, rsp_sstart, rsp_freq
    .ref    transfer, transfer_table_sz
    .global bridge_data, buff_clear_loop
    .def    ago_main, int_nmi

    .include "rx-dsp.h"
    .text

find_me             .ulong     0x6B28FE40
unit_designation    .uword     0  ; Unit number (Master = 0)
code_version        .string    "v1.0"
station_code        .string    "SS"

band_width    .string    "0750"

; Rotating center frequency table
; 32-bit values derived by mapping the sampling frequency (S) to
; the 0:2^32 range, then taking the ratio of center frequency
; (C) to S, i.e. C/S*2^32

cfreq_table:
    .word    0x01d2, 0xf1c9  ;  475 kHz
    .word    0x04b4, 0x39a7  ; 1225 kHz
    .word    0x0795, 0x8185  ; 1975 kHz
    .word    0x0a76, 0xc964  ; 2725 kHz
cfreq_table_end:
cfreq_table_sz    .set    cfreq_table_end-cfreq_table

cfreq_test    .word    0x0e66, 0x6758 ; 3750 KHz, used when
                                      ; trm_28 jumper is on

cfreq_toggle1    .set    0x0999  ; 2500 kHz
cfreq_toggle2    .set    0x9A3B


; Run constants
data_n          .set 512  ; Size of each FFT (# of IQ pairs)
data_discard    .set 512  ; number of words to discard from
                          ; Rx FIFO before taking data
data_minor_sz   .set 1    ; number of acquisitions per
                          ; half-buffer interrupt
abu_buff_sz     .set 1048 ; size of serial buffer
                          ; (2x major frame size IN BYTES)
fsync_sz        .set 4    ; # of serial frame sync bytes
                          ; (should be multiple of 4)
```

```
; Memory allocations
data_addr      .usect  ".data_v", 0x800, 1, 1
stackres       .usect  ".stack_v", 0x40, 1, 1
abu_buff_loc   .usect  ".sbuff_v", abu_buff_sz, 1, 1
abu_buff_hloc  .set    abu_buff_loc+abu_buff_sz/2  ; half-way

; Scratchpad RAM usage
bridge_count    .set    scratch
minor_count     .set    scratch+1
bspce_save      .set    scratch+2
bridge_size     .set    scratch+3
major_count     .set    scratch+4  ; two words!
nco_freq        .set    scratch+6  ; two words!
cfreq_tp        .set    scratch+8

    .bss    TempLmem,1*2,0,0  ;temporary dword


RXDSP_START
ago_main:

    rsbx    XF

; Processor setup
    ssbx    INTM         ; Disable interrupts
    stm     #(stackres+0x40), SP  ; set Stack Pointer
    stm     #npmst,PMST  ; Set processor mode/status
;    stm #defst0, ST0
;    stm    #defst1, ST1
    rsbx    SXM  ; Suppress sign extension
;    rsbx    XF
    nop          ; Space for branch to app
    nop

    ssbx    XF
    rpt     #64
    nop
    rsbx    XF

appcode:
;    stm    #0,state      ; Clear interrupt routine state
    stm     #0,AR0        ; Clear auxilliary register 0
    portw   AR0,wr_rs_rx  ; Reset AD6620 RSP
    portw   AR0,wr_disc   ; Enable parallel TLM drivers, I_Q out
    portw   AR0,wr_dog    ; Strobe watchdog timer
```

```
    stm    #0,AR0  ; Clear all auxiliary registers
    stm    #0,AR1
    stm    #0,AR2
    stm    #0,AR3
    stm    #0,AR4
    stm    #0,AR5
    stm    #0,AR6
    stm    #0,AR7

    stm    #0FFh,IFR  ; Clear any pending interrupts
    stm    #ntss,TCR  ; Stop timer, if running
    .global read_init
; Main data code start
read_init:
    call    rsp_reset
    call    rsp_init
    call    rsp_clear
    call    rsp_mstart

    rpt    #4444  ; Let the AD6620 do its first initialization in peace
    nop

    call    rsp_reset

    portr    rs_rx_fifo, AR0  ; Reset Rx FIFO - also wired to Slave
    nop
    nop

    stm    #lsb_sel, AR0  ; Reset acq_seq
    portw  AR0, wr_disc
    nop


    ; TSP shutoff

;     stm    #(tspc_Free+tspc_fsm+tspc_nXrst+tspc_nRrst), TSPC
    stm    #(tspc_Free+tspc_fsm), TSPC

    ; BSP prep

    stm    #(bspc_Free+bspc_fsm), BSPC0  ; reset BSP
    stm    #(int_bx), IMR  ; unmask serial transmit interrupt
    ; 10-bit words, enable tx autobuffer, halt after first half-buffer
    stm    #(bspce_fe+bspce_bxe+bspce_haltx), BSPCE0
    stm    #(abu_buff_loc-0x800), AXR  ; where in RAM does
                                       ; the tx buffer start?
    stm    #(abu_buff_sz), BKX  ; buffer size
```

```
; Clear entire serial buffer
    stm     #abu_buff_loc, AR4
    stm     #abu_buff_sz-1, BRC
    rptb    buff_init_loop - 1

    st    #0h, *AR4+
    .global buff_init_loop, head_ramp, major_loop
buff_init_loop:

; set initial frequency table position

    st    #cfreq_table_sz-2, @cfreq_tp

;    rsbx    INTM ; global interrupt enable

; All Aux Registers are fungible in the main loop: values which must
; be preserved over time are stored in the scratchpad RAM as defined
; above.  Note that only AR6 and AR7 are required to be preserved by
; the DSP Library functions (and most others), all other ARx may be
; modified within function calls.

    ld     #0, A
    dst    A, @major_count

; Set initial bspce_save such that the first acquisition will
; write to the second half of the serial buffer
    st    #bspce_xh, @bspce_save

    ; boot lag: insert >50 ms delay to allow
    ; everyone plenty of time to boot up
    stm    #16384, BRC
    nop
    nop
    rptbd    boot_delay_loop - 1
    nop
    nop

    nop

    rpt    #4096
    nop

    nop
    nop

boot_delay_loop:
```

```

    nop
    nop

    ; now synchronize units for the first time
    call    sync_units

    ; Start BSP transmits
    stm     #(bspc_Free + bspc_fsm + bspc_nXrst), BSPC0   ; hold fsm bit
    nop
    nop

;```````
; Main loop
;_______

major_loop:
    .global    major_loop

    ; Method:
    ;
    ; Master waits for acq_seq_rdy from Slaves, then raises
    ; acq_seq_out, starting synced acquisition.  Note any previous
    ; acquisition will still be transferring its last half-buffer,
    ; and the ABU will be set to halt transmissions when that half
    ; is done.  The acq_seq_rdy & out lines are both held high on
    ; All, until the end of this new acquisition and half-buffer
    ; fill.  Then Master waits for its own ABU haltx, and then for
    ; !acq_seq_rdy (signaling Slaves have hit ABU haltx), before
    ; dropping acq_seq_out, signaling time for the next
    ; synchronized ABU startup.

    call    cfreq_walk

    ; acquisition sync
    call    sync_units

    portr    rs_rx_fifo, AR0   ; Reset Rx FIFO - also wired to Slave

    cmpm    *(unit_designation), #0
    bc    slave_startup, NTC

    ; if Master (unit 0) insert delay time to allow
    ; Slaves time to detect acq_seq_out and start up
    rpt    #128
    nop

```

```
    call    rsp_mstart  ; start digitizing as Master

    b    data_acq_start

slave_startup:

    call    rsp_sstart  ; start digitizing as Slave

;
; Reads
;
data_acq_start:

    st    #0, @bridge_count  ; reset bridged data counter
    st    #0, @minor_count   ; reset minor frame counter

;
; Set wait states for Rx FIFO
;
    ldm   SWWSR, A
    ld    #65535, B
    xor   #7, #swwsr_is, B  ; (0b111<<Nset XOR 0d65535) creates bitmask
    and   B, A              ; mask out bits of interest
    or    #0, #swwsr_is, A  ; (A or Nwait<<Nset) to set Nwait to Nset
    ; Nwait = 0 means no additional waits
    stlm    A, SWWSR

    .global    pre_disc, pre_read
pre_disc:

; loop to read and discard first data out of AD6620
    stm    #(data_discard), AR2
    nop
rx_discard_loop:
    ; read only if rx fifo is nonempty
    portr    rd_disc, AR0
    nop
    nop
    bitf     AR0, rx_efo

    bc     discard_fifo_empty, NTC
    portr  rd_rx_out, AR1  ; read data into AR1
    mar    *AR2-           ; decrement word counter
discard_fifo_empty:

    banz    rx_discard_loop, *AR2

```

```

; loop to read data from Rx FIFO into RAM
    stm    #2, AR0
    stm    #data_addr, AR1  ; set address for first data word
    stm    #(data_n*2-1), AR2
pre_read:
    nop
rx_read_loop:
    ; read only if rx fifo is nonempty
    portr    rd_disc, AR3
    nop
    nop
    bitf     AR3, rx_efo

    bc    read_fifo_empty, NTC

    portr    rd_rx_out, *AR1+
    mar      *AR2-  ; decrement word counter

read_fifo_empty:

    banzd    rx_read_loop, *AR2
    nop
    nop

;
; Set wait states for other stuff (full 7)
;
    ldm    SWWSR, A
    or     #7, #swwsr_is, A ; (A or Nwait<<Nset) to set Nwait to Nset
    stlm   A, SWWSR
    nop
    nop

; Stop acquisition
    call    rsp_reset

    ; drop acq flags
    stm      #lsb_sel, AR0
    portw    AR0, wr_disc

;
; End data collection, begin serial data handling
;
    st    #4*data_n, @bridge_size

; entry point for bridging data transfers
```

```
; over multiple serial half-buffers
bridge_data:
    .global    bridge_data

    stm     #(lsb_sel), AR0  ; send acq_seq, set lsb_sel
    portw   AR0, wr_disc

    nop
    nop
    ; Determine serial buffer position
    stm     #abu_buff_loc, AR3  ; load buffer base

    bitf    @bspce_save, #bspce_xh  ; read XH out of
                                    ; stored BSPCE register
    nop
    nop
    bc      buff_skip, NTC  ; if first half _finished_
                            ; (XH=0, NTC), do nothing

    addm     #(abu_buff_sz/2), @AR3
    stm      #(lsb_sel), AR0
    portw    AR0, wr_disc

buff_skip:
    nop
    nop
    nop
    nop

abu_first_half:
    .global    abu_first_half

    ssbx    INTM

; Clear serial buffer half
    mvmm    AR3, AR4
    nop
    rpt     #(abu_buff_sz/2-1)
    st      #0x00, *AR4+


; reset AR4 for data copy
    mvmm    AR3, AR4
    nop
    nop

abu_fill_start:
```

```
    .global    abu_fill_start


    ; two-byte frame sync 0xEB90
    st    #0xEB, *AR4+
    st    #0x90, *AR4+

    ; two-byte infofoop
    ld     *(unit_designation), #6, A  ; byte 1, 2 MSB, unit number
    or     @minor_count, A  ; byte 1, 6 LSB, minor frame number
    stl    A, *AR4+
    dld    @major_count, A  ; byte 2, 8 bit, major frame number
    and    #0xFF, A
    stl    A, *AR4+

dinner_is_ready:
    .global dinner_is_ready

    ld    #((abu_buff_sz/2-fsync_sz)/2-1), B  ; let's stick the
                                               ; BRC value here...

    ; If haltx is true, we are starting a
    ; new major frame, so write the header
    bitf    BSPCE0, #bspce_haltx
    bc    header_skip, NTC

    call    hwrite  ; write out header, uses &
                    ; modified write address in AR4
    sub     #16, B  ; oops, 32 bytes less space in this half buffer

header_skip:
    ; copy raw data (words) into serial buffer (bytes)
    ld     #data_addr, A
    add    @bridge_count, #-1, A  ; div by 2 to increment
                                  ; input word-wise
    stlm    A, AR2  ; store a copy of the data to AR2
    stlm    B, BRC  ; and here's our BRC setup

    mvdm    @bridge_size, AR0   ; goal bridge size
    mvdm    @bridge_count, AR1  ; current count

    rptb    rawdata_loop - 1

    ld     *AR2, -8, A  ; load (data word) >> 8 to Acc
    and    #0xFF, A     ; mask to low-byte
    stl    A, *AR4+     ; save to serial buffer
    mar    *AR1+        ; count bytes
```

```
    ld      *AR2+, A      ; reload and increment
    and     #0xFF, A      ; mask
    stl     A, *AR4+      ; save
    mar     *AR1+         ; count bytes

    cmpr    LT, AR1   ; If we're not done with a bridged data sequence
    nop
    nop
    xc      1, NTC
    rsbx    BRAF
    nop
    nop
    nop
    nop
    nop
    .global    rawdata_loop, dbg_transfer_skip
rawdata_loop:

    nop
    nop

    mvmd    AR1, @bridge_count

; Bit reverse and add start/stop bits
    ldm     AR3, A
    ld      #(abu_buff_sz/2), B

    call    _serial_cook


;     rsbx     INTM

    addm     #1, @minor_count
    nop
    nop
    stm      #(lsb_sel), AR0
    portw    AR0, wr_disc

; buffer is now loaded
    call     rsp_clear  ; clear NCO RAM, do it here
                        ; since we have some free time

; Strobe watchdog- once per minor frame cycle
    stm       #0, AR0     ; Data is not used- just the wr_dog strobe
    portw     AR0,wr_dog  ; Strobe the watchdog

```

```
; clear interrupt flags, then idle until an interrupt.
    .global    pre_sleep
pre_sleep:

    ssbx    INTM
    stm     #0FFh,IFR  ; Clear any pending interrupts

    idle    2  ; and now...we wait.

    .global    post_sleep
post_sleep:

; check for aux int -> serial monitor
;     bitf    IFR, #int_3
;     cc      inth_3, TC

; make sure we had a serial interrupt
    bitf    IFR, #int_bx
    bc      pre_sleep, NTC  ; stray interrupt, go back to IDLE
    nop

    stm    #int_bx, IFR     ; clear int flag

    mvmd    BSPCE0, @bspce_save ; store control extension register

; if this is a new major frame, we need to sync
; everyone up by waiting for all ABU haltx.

    bitf    BSPCE0, #bspce_haltx
    bc      abu_restart_skip, NTC

abu_haltx_wait:
    .global    abu_haltx_wait

; ABU has halted.  Reset ABU, and synchronize
    stm    #(bspce_fe+bspce_bxe), BSPCE0  ; 10-bit words, enable
                                          ; tx ABU, disable haltx

    call    sync_units

; Start BSP transmits
    stm    #(bspc_Free + bspc_fsm + bspc_nXrst), BSPC0  ; hold fsm bit
    nop
    nop

    rpt    #16383
    nop
```

```

; unset acq_seq, keep lsb_sel
    stm      #lsb_sel, AR0
    portw    AR0, wr_disc

abu_restart_skip:
    .global    abu_restart_skip

    mvdm    @bridge_size, AR0   ; need to copy these
    mvdm    @bridge_count, AR1  ; to use CMPR
    nop
    nop
    cmpr    LT, AR1            ; If we're not done with
                               ; a bridged data sequence,
    bc      bridge_data, TC    ; jump to bridge_data to
                               ; continue transfers, otherwise...

    dld    @major_count, A     ; increment major frame counter
    add    #1, A
    dst    A, @major_count

; The final half is transmitting, we want to halt when it finishes.
    stm      #(lsb_sel), AR0
    portw    AR0, wr_disc
    orm      #bspce_haltx, BSPCE0
    nop
    nop
;    stm     #(lsb_sel), AR0
;    portw   AR0, wr_disc

    b    major_loop  ; new data acquisition

;```````
; Main acquisiton ('appcode') branch done
;_______

;
; Interrupts
;

; Non-Maskable Interrupt
;      this is hit by the watchdog
int_nmi:

    nop
    stm    #npmst, PMST  ; Reset PMST to be sure IPTR -> 0x80

```

```
; Alternative: IPTR=0x1FF, OVLY=1, all else =0
; This should set things up to completely reload
; the program from the EPROM on reset
;     stm    #0xFFA0, PMST

    reset  ; I don't have to take this.  ...I'm going home.

    ret  ; should never get here!


inth_3:
    ssbx    INTM
    stm     #int_3, IFR  ; clear int flag

    retd
    nop
    nop

;```````
; cfreq_walk
;
; walks through the table of center frequencies
; at label #cfreq_table
;
; stores table position @cfreq_tp and current NCO value
; as a 32-bit number @nco_freq

cfreq_walk:

    portr    rd_disc, AR0
    nop
    nop
    bitf     AR0, #trm_28
    ; If trm_28 is low (NTC, jumper on), skip the walk.
    bc       walk_skip, NTC

    addm    #2, @cfreq_tp
    nop
    nop
    cmpm    @cfreq_tp, #cfreq_table_sz
    nop
    nop
    xc    2, TC
    st    #0, @cfreq_tp

    nop
    nop
```

```

    ld      #(cfreq_table), A
    add     @cfreq_tp, A
    stlm    A, AR0

    b    cfreq_commit

walk_skip:
    ; load a single frequency instead
    stm    #(cfreq_test), AR0

cfreq_commit:

    nop
    nop

    ld      *AR0+, B
    stl     B, @nco_freq
    sftl    B, #8
    sftl    B, #8
    ld      *AR0, A
    stl     A, @(nco_freq+1)
    or      B, A

    nop
    nop

    stm    #acq_ant_1, AR0  ; enable antenna 1

    ; frequency is in A.  subtract toggle freq and save result to B
    sub    #cfreq_toggle1, #16, A, B
    sub    #cfreq_toggle2, B

    xc     2, BGEQ          ; if B>=0, cfreq >= toggle freq
    stm    #acq_ant_2, AR0  ; so enable antenna 2
    nop

    portw  AR0, wr_disc     ; write out antenna toggle lines

    call   rsp_freq

    retd
    nop
    nop

;```````
; sync_units
```

```
;
; synchronizes master/slave units by
; toggling and waiting for latched lines

sync_units:

; Raise test2 line.  On the Master this should do nothing (NC),
; on the Slaves it signals the Master they are ready.

    stm      #(acq_seq_rdy+lsb_sel), AR0
    portw    AR0, wr_disc
    nop
    nop

; Check status of TSP, wait for IN1 & IN2 high
ready_loop:

    nop

    bitf    TSPC, #tspc_in0
    bc      ready_loop, NTC

    bitf    TSPC, #tspc_in1
    bc      ready_loop, NTC

; Raise acq_seq_out--on the Master this signals the Slaves
; to start, on the Slaves it does nothing (NC).
    stm    #(acq_seq_out+acq_seq_rdy+lsb_sel), AR0
    portw    AR0, wr_disc

    retd
    nop
    nop

static_header:
    .word    0xFE, 0x6B, 0x28, 0x40
    .string    "RxDSP"
static_header_end:

static_header_len    .set    static_header_end-static_header-1
```

A.4.4 Sondrestrom Utilities

Below are two short Python utility scripts: the first acquires MI-ARC data from the serial port, saves it to disk, and will parse out major frames and save them separately for real-time display, if desired. The second is a mostly functional real-time display script which makes use of the Gnuplot.py module.

tridsp-acq.py

```python
#!/usr/bin/env python

# MCB 27 Sept. 2013. This code is the same as tridsp-acq.py except
# there are new lines (41-43) that power cycle the receiver by
# changing the state of the serial DTR line.

import serial,signal,sys,time
from struct import unpack,pack
from datetime import datetime
import numpy as np
#import Gnuplot
#from matplotlib import pyplot as plt
from optparse import OptionParser

parser = OptionParser(usage="""tridsp-acq.py: read in triple-DSP multiplexed serial data.""")

parser.set_defaults(verbose=False,port="/dev/ttyS0", ofp="SStridsp", nboards=3, majfsync=0xFE6B2840, majfsz=4, minfsz=524, limit=242936, filesync=0xb0f919c5025d, acqsz=4080, rtd=True, rtdfile="/tmp/trirtd.data")

parser.add_option("-o", "--outfile", type="str", dest="ofp", help="Output file prefix.  [default: %default]")
parser.add_option("-p", "--port", type="str", dest="port", help="Serial port number.  [%default]")
parser.add_option("-r", "--rtd", action="store_true", dest="rtd", help="Write out major frames for RTD. [%default]")
parser.add_option("-R", "--rtdfile", type="str", dest="rtdfile", help="File for RTD data. [%default]")
parser.add_option("-n", "--nboards", type="int", dest="nboards", help="Number of multiplexed DSP boards.  [%default]")
parser.add_option("-s", "--maj-sync", type="int", dest="majfsync", help="Major frame sync pattern.  [%default]")
parser.add_option("-z", "--maj-size", type="int", dest="majfsz", help="# of Minor frames per Major frame.  [%default]")
parser.add_option("-Z", "--min-size", type="int", dest="minfsz", help="# of bytes per Minor frame.  [%default]")
parser.add_option("-X", "--limit", type="int", dest="limit", help="Number of acquisitions to record.  [%default]")
parser.add_option("-v", "--verbose", action="store_true", dest="verbose",
                  help="print status messages to stdout.")

(o, args) = parser.parse_args()
```

```

o.majfsync = pack("4B", *[(o.majfsync>>i)&0xFF for i in [24,16,8,0]])
o.filesync = pack("6B", *[(o.filesync>>i)&0xFF for i in [40,32,24,16,8,0]])

try:
    inp = serial.Serial(port=o.port, baudrate=115200, bytesize=8,
        parity='N', stopbits=1)
except serial.SerialException:
        print "Unable to open serial port {0}.".format(o.port)
        sys.exit(1)

#Restore power to receiver by toggling the DTR line
time.sleep(5.0)
inp.setDTR(False)

print("Reading from serial port {0}...".format(o.port))

dtstr = datetime.today().strftime("%Y%m%d-%H%M%S")

ofn = "{0}-{1}.data".format(o.ofp, dtstr)
ofile = open(ofn, "w")

print("Taking {0} {1}-byte acquisitions ({2} hours).".format(o.limit,
    o.acqsz, o.limit*o.acqsz/11520/3600))

print("Writing data to {0}...".format(ofn))

if o.rtd:
        rfile = open(o.rtdfile, 'w')

print("Writing RTD data to {0}.".format(o.rtdfile))

framecount = 0
bytestr = ""
mfbsync = "".join([o.majfsync[i]*o.nboards for i in range(len(o.majfsync))])
mfbsize = o.nboards*o.majfsz*o.minfsz

running = True

acqcount = 0

while running and acqcount < o.limit:
        data = inp.read(o.acqsz)

        # build timestamp: 40-bit uint seconds since epoch, 24-bit uint
            microseconds
        timefl = time.time()
```

```
        timeint = int(timefl)
        timefrac = int((timefl-timeint)*(1e6))
        timestr = ( pack(">Q", timeint&0xFFFFFFFFFF)[3:] ) + ( pack(">I", timefrac)[1:] )

        ofile.write(o.filesync)
        ofile.write(pack('>H', framecount&0xFFFF))      # 16-bit counter
        ofile.write(timestr)
        ofile.write(data)

        framecount += 1

        # if we want rtd, add to bytestream, and if there's a major frame in there, write it

        if o.rtd:
                bytestr += data

                loc = bytestr.find(mfbsync)
                nloc = bytestr.find(mfbsync, loc+1)

                if (loc >= 0) and (nloc >= loc):
                        rfile.seek(0)
                        rfile.write(bytestr[loc-4 * o.nboards:nloc-4 * o.nboards])
                        rfile.flush()
#                       prelen = len(bytestr)
                        bytestr = bytestr[nloc-4*o.nboards:]
#                       postlen = len(bytestr)
#                       print("Wrote RTD file. {0} -> {1}".format(prelen,postlen))

        acqcount += 1

if o.rtd:
        rfile.close()
ofile.close()
```

tridsp-rtd.py

```
#!/usr/bin/env python
from os.path import getmtime
import sys
from struct import unpack,pack
from datetime import datetime
import numpy as np
import Gnuplot
```

```
#from matplotlib import pyplot as plt
from optparse import OptionParser
from math import ceil, floor

parser = OptionParser(usage="""tridsp-acq.py: read in triple-DSP multiplexed serial data.""")

parser.set_defaults(verbose=False, rfile="/tmp/trirtd.data", nboards=3,
                                                majfsync=0xFE6B2840, majfsz=4,
                                                minfsync=0xEB90, minfsz=524, dataplot=True)

parser.add_option("-r", "--rfile", type="str", dest="rfile", help="RTD data file.  [default: %default]")
parser.add_option("-n", "--nboards", type="int", dest="nboards", help="Number of multiplexed DSP boards.  [%default]")
parser.add_option("-s", "--maj-sync", type="int", dest="majfsync", help="Major frame sync pattern.  [%default]")
parser.add_option("-z", "--maj-size", type="int", dest="majfsz", help="# of Minor frames per Major frame.  [%default]")
parser.add_option("-S", "--min-sync", type="int", dest="minfsync", help="Minor frame sync pattern.  [%default]")
parser.add_option("-Z", "--min-size", type="int", dest="minfsz", help="# of bytes per Minor frame.  [%default]")
parser.add_option("-v", "--verbose", action="store_true", dest="verbose",
                help="print status messages to stdout.")
parser.add_option("-c", "--channel", action = "store",type = 'int', dest="chan_num", help = "channel number to display")
parser.add_option("-f", "--freq", action = "store_true", dest = "spectra")
parser.add_option("-b", "--band", action = "store", dest = "bw", type="float")
(o, args) = parser.parse_args()
def     b2iq(indat):
        # input : binary string containing 16-bit big-endian signed I/Q data
        # output: [[I array], [Q array]]

        num = unpack(">"+str(len(indat)/2)+"h", indat)
        return np.reshape(num, (2,-1), "F")

cplotd = {}
blist = range(o.nboards)

infile = open(o.rfile, 'r')
freqplot=Gnuplot.Gnuplot()
#freqplot("set title \"Unit {0}\"".format(k,cfreq))
freqplot("set term x11 noraise")
```

```
freqplot("set yrange [0:140]")
freqplot("set xrange [100:3100]")
freqplot("set style data lines")
#freqplot("set title \"{0} KHz\"".format(cfreq))

oldtime = 0
first_loop = 0
plots= []

running = True

window = np.hanning(512)

while running:

        while True:
                newtime = getmtime(o.rfile)
                if newtime != oldtime:
                        break

        oldtime = newtime
        badness = False

        infile.seek(0)
        data = infile.read()

        # parse major frames and plot
        unitdata = [[]]*o.nboards

        for i in blist:
                bdata = data[i::o.nboards]
                majfr = "".join([bdata[j*o.minfsz+4:(j+1)*o.minfsz] for j in range(o.majfsz)]) # extract major frame
                minsyncs = "".join([bdata[j*o.minfsz:j*o.minfsz+4] for j in range(o.majfsz)])  #extract minor frame syncs

                minunit = [(x & 0xC0)>>6 for x in unpack(">4B", minsyncs[2::4])]
                if len(np.unique(minunit)) != 1:
                        print("Minor frame Unit number mismatch.")
                        badness = True

                minmajN = unpack(">4B", minsyncs[3::4])
                if len(np.unique(minmajN)) != 1:
                        print("Minor frames' major frame # mismatched.")
                        badness = True

```

```
            unit = unpack("B", majfr[9])[0]-0x30
            if unit != np.unique(minunit)[0]:
                print("Major frame doesn't match minor frame Unit #.")
                badness = True

            majN = unpack(">I", majfr[24:28])[0]&0xFF
            if majN&0xFF != np.unique(minmajN)[0]:  # mask to one-byte to test against minor frame #
                print("Major frame # doesn't match minor frames' major frame #.")
                badness = True

            # looks like a good major frame
            cfreq = int(round(unpack(">I", majfr[16:20])[0]/2.0**32*66666.6))
            nums = unpack(">"+str(len(majfr)/2-16)+"h", majfr[32:])

            unitdata[unit] = {      'cfreq' : cfreq,
                                    'eyes' : nums[::2],
                                    'ques' : nums[1::2] }

        if len(np.unique([ x['cfreq'] for x in unitdata ])) != 1:
            print("Center frequency mismatch!")

        if badness:
            continue

        cfreq = unitdata[0]['cfreq']

        if o.spectra:
            k = o.chan_num
            dfdb = o.bw/512.0
            freqlist = [cfreq-(o.bw/2.0) + dfdb * i for i in range(512)]
            eyes =  unitdata[k]['eyes']
            ques =   unitdata[k]['ques']
            fftdata = [eyes[j] + ques[j] * 1j for j in range(len(eyes))]

            spec = [10.0*y for y in np.log10([abs(x)**2 for x in np.fft.fft(window*fftdata,n=512,)])]       # power spectra
            spec = spec[len(spec)/2:]+spec[:len(spec)/2]    # swap from normal order

            if ((cfreq == 475) or (cfreq == 3750)) and (len(plots) > 0):
                freqplot.plot(*plots)
                plots = []
            plots.append(Gnuplot.Data(freqlist, spec))
```

Appendix B

Test-Particle Simulation, Distribution Building, and Growth-Rate Calculation Codes

Note that most of the simulation and growth rate Matlab codes were initially tested and debugged on a Core i5-4590S (4-core, 3 GHz), with 32 GB of RAM. So, not a powerhouse FLOPS-wise, but the code is pretty carefree regarding memory usage. Caveat emptor if you run this on a system with less RAM—Matlab may cry out, and forfeit.

B.1 Mirror Shards

Below are the primary test particle codes as used in this thesis. Historically, the mirror code was a single particle code provided by Dr. Wayne Scales. When moving towards parallelism, it was rewritten to be monolithic, and entirely run via the Matlab Distributed Compute Server (DCS) on GPU nodes. Then the computation was found to be entirely FLOPS-dependent, and running one or only a few particles on a single fast CPU core to be preferable to GPU massive parallelism. This requires as many CPU cores as possible, far beyond the artificially limited (due to Mathworks' prohibitive licensing fees) DCS max core number.

So, the decision was made to break the code apart into separate pieces, each of which would run be queued and run as a separate job on a PBS/TORQUE cluster compute system. The 'mirror shard' codes are so-named because they break the mirror simulation particles up among some number of 'shards', with each shard assigned a set number of calculation cores. A given core can work on one or many particles, and one or multiple shards can run on a given node—whatever makes for the best queuing setup.

The three primary codes are the distribute code, the Alice code, and the gather code.

B.1.1 Distribute

This code provides a Matlab function, `mirror_shards_distribute(n_run, n_shards)`, which generates the 'test distribution' with a given range of positions, velocities (given in eV), pitch angles, and azimuthal angles, and breaks it up among the specified number of shards. It splits the distribution up among a set of files of form mshard-r<n_run>-<i>of<n_shards>-input.mat, and also saves all pertinent distribution input parameters in the file 'mshards-r<n_run>-master.mat'.

It does the splitting in an excessively lazy manner, using Matlab's built-in `distributed()` function over the Distributed Compute Server (DCS). This requires that a core for each shard, i.e. the maximum number of shards possible is equal to the maximum number of cluster cores available for use with the DCS.

This splitting is quite frankly a holdover from the rushed transition from a monolithic design to the sharded design. With some work this code could be done away with entirely—the inline distribution-building function is very fast, so it could be done within each individual job, based on the input parameters given and the job's ID number. Then the only remaining function of this code would be to build the 'master' file used to save the input parameters.

```
function ret = mirror_shards_distribute(n_run, n_shards)
% mirror_shards_distribute()
%
% Breaks a data array down into a number of shards, for use on single-node
% local worker pools, so we don't have to deal with Matlab DCE
% limitations on cores.
% Feed it a run number, and the number of shards to break the data over.
% The fundamentals of the simulation are all set here.

    q = 1;
    m = 1;
    nt = 10000;   % # timesteps
    dt = .01;     % step length
    qE = 0;
    qmt2 = q/m*dt/2;

    B0 = 50e-6; % Magnetic field base is 50 uT
    v0 = 0.00989179273; % likewise velocity base
                        % in PSL is equivalent to 25 eV
    r0 = 0.337212985; % based on Larmour radius w/ above,
                      % length base is ~0.337 m
    t0 = 7.14477319e-7; % based on B, Larmour period ~714 ns in s

    target_length = 5000;   % in km
    target_z = -target_length*1000/r0;  % negative because we're
                                        % launching upwards
    long_enough = 1000000000;
    mirror_ratio = 5;
    saved_steps = 1000;
```

```
% So Bsim=Breal/50uT, vsim=vreal/25 eV, and xsim=xreal/0.337m
% So a 100x100x1000 simulation extent is a 33.7x33.7x337m volume
% So dt ~71.4ns, and 1000 timesteps is 71us

% assumes 'end point' is z=0
length_factor = target_z^2/(mirror_ratio-1);

x_range = 0;
y_range = 0;
z_range = target_z;
v_range = [ 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, ...
    256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, ...
    784, 841, 900, 961, 1024, 1089, 1156, 1225 ]; % linear in v
t_dphi = 3*pi/256; % delta for co-latitude
t_domega = 0.001; % delta for solid angle in steradians
%p_range = 0:pi/7:pi; %0:pi/15:pi/2;

v_distrib = build_distrib(v0, x_range, y_range, z_range, v_range,
   t_dphi, t_domega);

% Re-run particles that failed due to max timestep limit in Run 4
load tzind.mat t_zind
v_distrib = v_distrib(:,t_zind)

parpool('torque_4nodes',n_shards)

N_part = size(v_distrib,2);
disp([ 'Distributing ' num2str(N_part) ' particles over '
   num2str(n_shards) ' node shards...' ])

v_sdivdist = distributed(v_distrib);

disp('Start')
tic

% spmd (single program, multiple data) is a more generalized
% multithreaded methodology than parfor, and allows use of
% distributed/codistributed functionality to split up arrays
spmd

    v_localdist = getLocalPart(v_sdivdist);
    N_dpart = size(v_localdist, 2);
    chunk_inds = globalIndices(v_sdivdist,2);

    disp( [ 'Shard ' num2str(labindex) ': ' num2str(N_dpart) ' particles
       (' num2str(chunk_inds(1)) ':' num2str(chunk_inds(end)) ').' ] )
```

```
        % Have to call a function to use save inside an spmd.
        % Because...raisins.
        % Local workspace memory separation something something.
        save_mah_data_plz(n_run, labindex, n_shards, N_dpart, chunk_inds, v_localdist);

    end

    toc
    disp('Done.')

    save(['mshards-r' num2str(n_run) '-master.mat'], ...
        'n_run', 'n_shards', 'N_part', 'v_distrib', ...
        'q', 'm', 'nt', 'dt', 'qE', 'qmt2', ...
        'B0', 'v0', 'r0', 't0', 'target_length', 'target_z', ...
        'long_enough', 'length_factor', 'mirror_ratio', ...
        'saved_steps', 'v_range', 't_dphi');

    ret = 0;

end

function save_mah_data_plz(n_run, labindex, n_shards, N_shardpart, chunk_inds, v_sharddist)
    % Just a function to save data.  Raisins.

    save(['mshard-r' num2str(n_run) '-' num2str(labindex) 'of' num2str(n_shards) '-input.mat'], ...
        'n_run', 'N_shardpart', 'chunk_inds', 'v_sharddist');

end

function d = build_distrib(v0, x_range, y_range, z_range, v_range, t_dphi, t_domega)
    % Build particle distribution

    % initial positions x y z
    % initial velocities v theta phi (magnitude, azimuth, co-latitude)
    %   mag 25:2000 eV, azi 0, el 0:pi/2
    % input as [ x y z v theta phi ] columns in v_distrib_raw

    t_phis = 0+t_dphi:t_dphi:pi/2-t_dphi; % range of phis, discard first (pole) and last (plane)

    angle_list = [ 0 0 ];
    for i=1:length(t_phis)
```

```
        t_phi = t_phis(i);
        angle_list = [ angle_list ; 0 t_phi ];
    end

    %angle_list = angle_list([ 1 2 3 4 19 20 21 22 38 39 40 41 ],:)

    % limit angles for tests
    %angle_list = angle_list(sin(4*angle_list(:,2)).^2 >= 0.995,:); % wedges in azimuthal angle

    v_distrib_raw = zeros(6,length(x_range) * length(y_range) * length(z_range) * length(v_range) * length(angle_list));
    vdr_ind = 1;
    for i=1:length(x_range)
        for j=1:length(y_range)
            for k=1:length(z_range)
                for l=1:length(angle_list)
                    for m=1:length(v_range)
                        v_distrib_raw(:,vdr_ind) = [ x_range(i) y_range(j) z_range(k) v_range(m) angle_list(l,1) angle_list(l,2) ];
                        vdr_ind = vdr_ind + 1;
                    end
                end
            end
        end
    end

    % Transform v_mag,theta,phi to v_x, v_y, v_z
    v_distrib = v_distrib_raw;
    t_v = sqrt(3.913903e-6*v_distrib_raw(4,:))/v0;
    % number is 2/(m_e*c^2) in eV^-1
    v_distrib(4,:) = t_v .* cos(v_distrib_raw(5,:)) .* sin(v_distrib_raw(6,:));
    v_distrib(5,:) = t_v .* sin(v_distrib_raw(5,:)) .* sin(v_distrib_raw(6,:));
    v_distrib(6,:) = t_v .* cos(v_distrib_raw(6,:));

    d = v_distrib;
end
```

B.1.2 Alice

The Alice code provides `mirror_shards_alice(n_shard, n_cores, master_file)`, the primary worker function of the system. It must be fed its shard number and the assigned number of cores by its calling PBS script. An incorrect core number will result in baffling failures to run. It loads its distribution from the input file corresponding to its `n_shard`, performs the processing until particles reach their target altitude, then saves its results as codistributed arrays to ‘`mshard-r<n_run>-<n_shard>of<n_shards>-output.mat`’.

```
function ret = mirror_shards_alice(n_shard, n_cores, master_file)
% mirror_shards_alice()
%
% Does the actual simulation work in the mirror_shards_* system.

    p_g = load(master_file,'n_run','n_shards', ...
        'dt','qE','qmt2','r0', 'long_enough', ...
        'target_length','mirror_ratio', 'saved_steps');

    target_z = -p_g.target_length*1000/p_g.r0;
    % negative because we're launching upwards
    length_factor = target_z^2/(p_g.mirror_ratio-1);
    % assumes 'end point' is z=0

    % load values from p_g
    n_run = p_g.n_run; n_shards = p_g.n_shards; dt = p_g.dt; qE = p_g.qE;
    qmt2 = p_g.qmt2; long_enough = p_g.long_enough; saved_steps = p_g.saved_steps;

    % n_run, N_shardpart, chunk_inds, v_sharddist;
    p_s = load(['mshard-r' num2str(n_run) '-' num2str(n_shard) 'of' num2str(n_shards) '-input.mat'], ...
        'N_shardpart','v_sharddist');

    N_shardpart = p_s.N_shardpart;
    v_sharddist = p_s.v_sharddist;

    parpool('local', n_cores);

    % This is local to the shard now, but we'll just redefine below for the
    % part of the distribution that's local to each worker.
    v_sdivdist = distributed(v_sharddist);

    disp([ 'Simulating ' num2str(N_shardpart) ' particles over INFINITE timesteps...' ])
    tic
    disp('Start')

    % spmd (single program, multiple data) is a more generalized
    % multithreaded methodology than parfor, and allows use of
    % distributed/codistributed functionality to split up arrays
    spmd

```

```
        v_localdist = getLocalPart(v_sdivdist);
        N_dpart = size(v_localdist, 2);
        chunk_inds = globalIndices(v_sdivdist,2);

        disp( [ 'Running ' num2str(N_dpart) ' particles ('
            num2str(chunk_inds(1)) ':' num2str(chunk_inds(end)) ') in Lab '
            num2str(labindex) '.' ] )

        % Pre-allocate result arrays
        gm_X = zeros([ 3, saved_steps, N_dpart ], 'double'); % x,y,z
        gm_V = zeros([ 3, saved_steps, N_dpart ], 'double'); % vx,vy,vz
        gm_Bv = zeros([ 3, saved_steps, N_dpart ], 'double'); % Bx,By,Bz

        % redundant array of results, seven 3-vectors containing
        % 1,2 position and velocity at target-z (z_t)
        % 3, # of timestep before z_t, after, and actual calculated crossing
            time
        % 4,5 position and velocity of pre-z_t timestep
        % 6,7 position and velocity of post-z_t timestep
        gm_result = zeros([ 3, 7, N_dpart ], 'double');

        active_indices = 1:N_dpart;

        % Get B at initial positions
        % permute() lets us slot a (3,N) data peg into a (3,M,N) hole
        gm_X(:,end-1,:) = permute(v_localdist(1:3,:),[1 3 2]);
        gm_V(:,end-1,:) = permute(v_localdist(4:6,:),[1 3 2]);

        % Recall all arrays are (dimension, timestep, particles)
        gm_B_x = squeeze(-gm_X(1,end-1,active_indices) .*
            gm_X(3,end-1,active_indices) / length_factor);
        gm_B_y = squeeze(-gm_X(2,end-1,active_indices) .*
            gm_X(3,end-1,active_indices) / length_factor);
        gm_B_z = squeeze(1+gm_X(3,end-1,active_indices).^2 / length_factor);

        % Calculate 2nd position with Boris Mover
        gm_v_mh = squeeze(gm_V(:,end-1,:));
        gm_v_minus = gm_v_mh + qE;

        gm_B = [ gm_B_x gm_B_y gm_B_z ].';
        gm_Bv(:,end,:) = gm_B;
        gm_t_vec = qmt2*gm_B;
        gm_s_vec = 2*gm_t_vec./(1+gm_t_vec.^2);
        gm_v_prime = gm_v_minus + cross(gm_v_minus,gm_t_vec,1);
        gm_v_plus = gm_v_minus + cross(gm_v_prime,gm_s_vec,1);

        gm_V(:,end,:) = 0.5 .* (gm_v_mh + gm_v_plus + qE);
```

```
        gm_X(:,end,:) = gm_X(:,end-1,:) + gm_V(:,end-1,:) .* dt;

        tstep = 1;
        % Loop until all particles are done.
        while ~isempty(active_indices)
            tstep = tstep + 1;

            % shift saved-data matrices down one row
            gm_X(:,1:end-1,active_indices) = gm_X(:,2:end,active_indices);
            gm_V(:,1:end-1,active_indices) = gm_V(:,2:end,active_indices);
            gm_Bv(:,1:end-1,active_indices) = gm_Bv(:,2:end,active_indices);

            if labindex == 1 && mod(tstep,10000) == 0
                display(['Step ' num2str(tstep) ', '
                    num2str(length(active_indices)) ...
                    ' particles active, min/max z = '
                        num2str(min(gm_X(3,end,active_indices))) '/'
                        num2str(max(gm_X(3,end,active_indices))) '.'])
            end

            % Recall all arrays are (dimension, timestep, particles)
            gm_B_x = squeeze(-gm_X(1,end-1,active_indices) .*
                gm_X(3,end-1,active_indices) / length_factor);
            gm_B_y = squeeze(-gm_X(2,end-1,active_indices) .*
                gm_X(3,end-1,active_indices) / length_factor);
            gm_B_z = squeeze(1+gm_X(3,end-1,active_indices).^2 /
                length_factor);

            % half-step due to E-field
            gm_v_minus = squeeze(gm_V(:,end-1,active_indices) + qE);

            gm_B = [ gm_B_x gm_B_y gm_B_z ].';
            gm_Bv(:,end,active_indices) = gm_B;
            gm_t_vec = qmt2*gm_B;
            gm_s_vec = 2*gm_t_vec./(1+gm_t_vec.^2);
            % these calculate the B-field effects
            gm_v_prime = gm_v_minus + cross(gm_v_minus,gm_t_vec);
            gm_v_plus = gm_v_minus + cross(gm_v_prime,gm_s_vec);

            % second half-step from E-field
            gm_V(:,end,active_indices) = gm_v_plus + qE;

            gm_X(:,end,active_indices) = gm_X(:,end-1,active_indices) +
                gm_V(:,end-1,active_indices) .* dt;

            % check if next z-pos passes the target plane z=0
            strike_indices = active_indices(gm_X(3, end, active_indices) >=
```

```
                0);
            if ~isempty(strike_indices)
                %display([ 'Timestep ' num2str(tstep) ': '
                    num2str(length(strike_indices)) ' strikes.' ]);
                % interpolate absolute strike XVT
                % NB: assumes 'target z' is z=0 plane
                t_nStrikes = length(strike_indices);
                t_V0 = squeeze(gm_V(:,end-1,strike_indices)); % init and
                t_V1 = squeeze(gm_V(:,end,strike_indices)); % final vel
                t_X0 = squeeze(gm_X(:,end-1,strike_indices)); % init and
                t_X1 = squeeze(gm_X(:,end,strike_indices)); % final pos

                % acceleration from x_pre to x_post
                t_a01 = ( t_V1-t_V0 )/dt;

                % time to z=0
                t_t0t = ( -t_V0(3,:) + sqrt(t_V0(3,:).^2 -
                    2*t_a01(3,:).*t_X0(3,:)) )./t_a01(3,:);
                % velocity at z=0
                t_Vt = t_V0 + bsxfun(@times,t_a01,t_t0t);
                % complete pos at z=0
                t_Xt = t_X0 + bsxfun(@times,t_V0,t_t0t) +
                    0.5*bsxfun(@times,t_a01,t_t0t.^2);

                % target x, target v, times, x0, v0, x1, v1
                gm_result(:,1,strike_indices) = squeeze(t_Xt);
                gm_result(:,2,strike_indices) = squeeze(t_Vt);

                t_t = [ (tstep+t_t0t)*dt ; repmat(tstep,1,t_nStrikes) ;
                    t_t0t ];

                gm_result(:,3,strike_indices) = t_t;

                gm_result(:,4,strike_indices) = squeeze(t_X0);
                gm_result(:,5,strike_indices) = squeeze(t_V0);

                gm_result(:,6,strike_indices) = squeeze(t_X1);
                gm_result(:,7,strike_indices) = squeeze(t_V1);

                active_indices = active_indices( ~ismember(active_indices,
                    strike_indices) );
            end

        end

        display(['Final timesteps: ' num2str(tstep) '.'])
```

```
        t_codist_result = codistributor1d(3, codistributor1d.unsetPartition,
            [3, 7, N_shardpart]);
        t_codist_saved = codistributor1d(3, codistributor1d.unsetPartition,
            [3, saved_steps, N_shardpart]);

        % build codist arrays
        r_divres = codistributed.build(gm_result, t_codist_result,
            'noCommunication');
        r_divsavX = codistributed.build(gm_X, t_codist_saved,
            'noCommunication');
        r_divsavV = codistributed.build(gm_V, t_codist_saved,
            'noCommunication');
        r_divsavB = codistributed.build(gm_Bv, t_codist_saved,
            'noCommunication');

    end % spmd

    % gather() to recombine distributed arrays
    r_shard_res = gather(r_divres);
    r_shard_X = gather(r_divsavX);
    r_shard_V = gather(r_divsavV);
    r_shard_B = gather(r_divsavB);
    r_shard_dist = gather(v_sdivdist);

    save(['mshard-r' num2str(n_run) '-' num2str(n_shard) 'of'
       num2str(n_shards) '-output.mat'], ...
        'N_shardpart', 'r_shard_dist', ...
        'r_shard_res', 'r_shard_X', 'r_shard_V', 'r_shard_B');

    disp('End')
    toc

    ret = 0;

end
```

B.1.3 Gather

Our final code provides `mirror_shards_gather(master_file)`, which requires only the 'master' file generated by the Distribute function. It loads all applicable output files, runs `gather()` to join the `distributed()` matrix, and saves the final results to '`mshards-r<n_run>-final.mat`'.

```
function mirror_shards_gather(master_file)
% mirror_shards_gather()
%
% Compiles the output produced by per-node mirror shards that ran on input
```

```
% constructed by mirror_shards_distribute().

    % load the things we care about
    p_g = load(master_file, ...
        'n_run', 'n_shards', 'N_part', 'v_distrib', 'saved_steps');


    n_run = p_g.n_run; n_shards = p_g.n_shards; saved_steps =
        p_g.saved_steps;
    N_part = p_g.N_part; v_distrib = p_g.v_distrib;

    parpool('torque_4nodes',n_shards);

    disp([ 'Loading ' num2str(n_shards) ' shard outputs...' ])

    spmd

        [ v_sharddist, gm_result, gm_X, gm_V, gm_B ] =
            load_mah_data_plz(n_run, labindex, n_shards);

        t_codist_distrib = codistributor1d(2,
            codistributor1d.unsetPartition, [6, N_part]);
        t_codist_result = codistributor1d(3, codistributor1d.unsetPartition,
            [3, 7, N_part]);
        t_codist_saved = codistributor1d(3, codistributor1d.unsetPartition,
            [3, saved_steps, N_part]);

        % gather() to copy from GPU RAM to Main Memory
        % ...or just to combine sharded data...
        r_divdist = codistributed.build(v_sharddist, t_codist_distrib,
            'noCommunication');
        r_divres = codistributed.build(gm_result, t_codist_result,
            'noCommunication');
        r_divsavX = codistributed.build(gm_X, t_codist_saved,
            'noCommunication');
        r_divsavV = codistributed.build(gm_V, t_codist_saved,
            'noCommunication');
        r_divsavB = codistributed.build(gm_B, t_codist_saved,
            'noCommunication');

    end

    disp('Done, saving...')

    r_dist = gather(r_divdist);
    r_res = gather(r_divres);
    r_savX = gather(r_divsavX);
```

```
    r_savV = gather(r_divsavV);
    r_savB = gather(r_divsavB);

    if ~isequal(r_dist, v_distrib)
        disp('Rebuilt distribution does not equal OG distribution from master file!')
    end

    save([ 'mshards-r' num2str(n_run) '-final.mat' ], ...
        'n_run', 'n_shards', 'N_part', 'v_distrib', ...
        'r_dist', 'r_res', 'r_savX', 'r_savV', 'r_savB');

    disp('...great success?')

end

function [ l_dist, l_res, l_gm_X, l_gm_V, l_gm_B ] = load_mah_data_plz(i_n_run,labindex,n_shards)

    p_d = load([ 'mshard-r' num2str(i_n_run) '-' num2str(labindex) 'of' num2str(n_shards) '-output.mat' ], ...
        'r_shard_dist', 'r_shard_res', 'r_shard_X', 'r_shard_V', 'r_shard_B');

    l_dist = p_d.r_shard_dist;
    l_res  = p_d.r_shard_res;
    l_gm_X = p_d.r_shard_X;
    l_gm_V = p_d.r_shard_V;
    l_gm_B = p_d.r_shard_B;

end
```

B.1.4 Bonus Code: GPU-Node Support

This is the final version of mirror code which runs on GPUs. Note that this is old, and there may be bug fixes and stuff in the Shards code that were not implemented here.

Algorithmically the code is essentially the same as Mirror Shards, but allocates its arrays with the 'gpuArray' parameter, and makes heavy use of `arrayfun()` on included functions, as this is faster on GPUs. As far as I can tell, when Matlab first sees an `arrayfun()` working on data stored in the GPU's RAM, it builds a CUDA kernel for that function, so future runs of the same type (as in a for loop) are GPU accelerated/parallelized as best as possible.

There is some turning point, dependent on number of particles and the necessary timesteps for the desired simulation length, between which either CPU sharding or GPU parallelism is the best choice. Of course, it also depends on what GPUs and CPUs are available.

mirror_gpu.m

```
function [ dist, resXVT, savedX, savedV ] = mirror_rtp()
% mirror_gpu_scriptable()
% Externally-scriptable version of test-particles-in-a-mirror-B-field
% simulation.  Comes with functions (below) to build distribution and
% construct the field, as well as various support functions.
% By default, will fall back to CPU processing if compatible GPUs
% are not present.

    q = 1;
    m = 1;
    nt = 10000;    % # timesteps
    dt = .1;     % step length
    qE = 0;
    qmt2 = q/m*dt/2;

    B0 = 1; % Magnetic field base is 50 uT
    v0 = 0.00989179273; % likewise velocity base in
                        % PSL is equivalent to 25 eV
    r0 = 0.337212985; % based on Larmour radius w/
                      % above, length base is ~0.337 m
    t0 = 0.714477319; % based on B, Larmour period ~714 ns

    target_length = 5000;    % in km
    target_z = -target_length/r0;  % negative because
                                   % we're launching upwards
    long_enough = 500;
    mirror_ratio = 5;
    saved_steps = 500;

    % So Bsim=Breal/50uT, vsim=vreal/25 eV, and xsim=xreal/0.337m
    % So a 100x100x1000 simulation extent is a 33.7x33.7x337m volume
    % So dt ~71.4ns, and 1000 timesteps is 71us

    % assumes 'end point' is z=0
    length_factor = target_z^2/(mirror_ratio-1);

    x_range = 0;
    y_range = 0;
    z_range = target_z;
    v_range = [25 484 1125];%[25, 36, 49, 64, 81, 100, 121, 144, ...
    % 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, ...
    % 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, ...
    % 1225]; % linear in v
    t_dtheta = 3*pi/256; % delta for co-latitude
    t_domega = 0.001; % delta for solid angle in steradians
    %p_range = 0:pi/7:pi; %0:pi/15:pi/2;
```

```
    v_distrib = build_distrib(v0, x_range, y_range, z_range, v_range,
        t_dtheta, t_domega);

    N_part = size(v_distrib,2);
    [ 'Simulating ' num2str(N_part) ' particles over maximum '
        num2str(long_enough) ' timesteps...' ]
    N_ts = nt+2;

    % distributed() is dumb, and requires the
    % chunking dimension to be the last one.
    v_sdivdist = distributed(v_distrib);

    disp('Start')
    tic

    % spmd (single program, multiple data) is a more generalized
    % multithreaded methodology than parfor, and allows use of
    % distributed/codistributed functionality to split up arrays
    spmd

        v_localdist = getLocalPart(v_sdivdist);
        N_dpart = size(v_localdist, 2);
        chunk_inds = globalIndices(v_sdivdist,2);

        d = gpuDevice();
        disp( [ 'Running ' num2str(N_dpart) ' particles ('
            num2str(chunk_inds(1)) ':' num2str(chunk_inds(end)) ') in Lab '
            num2str(labindex) ' on GPU ' num2str(d.Index) '.' ] )

        % Pre-allocate result arrays on GPU
        gm_X = nan([ 3, saved_steps, N_dpart ], 'double', 'gpuArray');
        gm_V = nan([ 3, saved_steps, N_dpart ], 'double', 'gpuArray');

        % result is x,y,z,vx,vy,vz,t,ts1,ts2
        gm_result = zeros([ 3, 3, N_dpart ], 'double', 'gpuArray');

        d = gpuDevice();
        t_tmem = d.TotalMemory;
        t_umem = t_tmem-d.AvailableMemory;
        disp( [ 'Memory Used: ' num2str(t_umem/1e9) '/' num2str(t_tmem/1e9)
            'GB (' num2str(t_umem/t_tmem*100) '%)' ]);

        active_indices = 1:N_dpart;
        length(active_indices)

        % Get B at initial positions
```

```
        % permute() lets us slot a (3,N) data peg into a (3,M,N) hole
        gm_X(:,end-1,:) = permute(v_localdist(1:3,:),[1 3 2]);
        gm_V(:,end-1,:) = permute(v_localdist(4:6,:),[1 3 2]);

        % Recall all arrays are (dimension, timestep, particles)
        gm_B_x = squeeze(arrayfun(@bxcalc,gm_X(1,end-1,:), gm_X(3,end-1,:),
            length_factor));
        gm_B_y = squeeze(arrayfun(@bycalc,gm_X(2,end-1,:), gm_X(3,end-1,:),
            length_factor));
        gm_B_z = squeeze(arrayfun(@bzcalc,gm_X(3,end-1,:), length_factor));

        % Calculate 2nd position with Boris Mover
        gm_v_mh = squeeze(gm_V(:,end-1,:));
        gm_v_minus = gm_v_mh + qE;

        gm_t_vec = tcalc(gm_B_x,gm_B_y,gm_B_z,qmt2);
        gm_s_vec = scalc(gm_t_vec);
        size(gm_v_minus)
        size(gm_t_vec)
        gm_v_prime = gm_v_minus + cross(gm_v_minus,gm_t_vec,1);
        gm_v_plus = gm_v_minus + cross(gm_v_prime,gm_s_vec,1);

        gm_V(:,end,:) = 0.5 .* (gm_v_mh + gm_v_plus + qE);
        gm_X(:,end,:) = gm_X(:,end-1,:) + gm_V(:,end-1,:) .* dt;

        tstep = 1;
        % Loop until all particles are done, or we've
        % done an absurd number of timesteps.
        while ~isempty(active_indices) && (tstep <= long_enough)
            tstep = tstep + 1;

            % shift saved-data matrices down one row
            gm_X(:,1:end-1,:) = gm_X(:,2:end,:);
            gm_V(:,1:end-1,:) = gm_V(:,2:end,:);

            if labindex == 1 && mod(tstep,100) == 0
                display(['Step ' num2str(tstep) ', '
                    num2str(length(active_indices)) ...
                     ' particles active, min/max z = '
                        num2str(min(gm_X(3,end-1,active_indices))) '/'
                        num2str(max(gm_X(3,end-1,active_indices))) '.'])
            end

            % Recall all arrays are (dimension, timestep, particles)
            gm_B_x = squeeze(arrayfun(@bxcalc, gm_X(1,end-1,active_indices),
                gm_X(3,end-1,active_indices), length_factor));
            gm_B_y = squeeze(arrayfun(@bycalc, gm_X(2,end-1,active_indices),
```

```
            gm_X(3,end-1,active_indices), length_factor));
        gm_B_z = squeeze(arrayfun(@bzcalc, gm_X(3,end-1,active_indices), length_factor));

        % half-step due to E-field
        gm_v_minus = squeeze(gm_V(:,end-1,active_indices) + qE);

        gm_t_vec = tcalc(gm_B_x,gm_B_y,gm_B_z,qmt2);
        gm_s_vec = scalc(gm_t_vec);
        % these calculate the B-field effects
        gm_v_prime = gm_v_minus + cross(gm_v_minus,gm_t_vec);
        gm_v_plus = gm_v_minus + cross(gm_v_prime,gm_s_vec);

        % second half-step from E-field
        gm_V(:,end,active_indices) = gm_v_plus + qE;
        gm_X(:,end,active_indices) = gm_X(:,end-1,active_indices) + gm_V(:,end-1,active_indices) .* dt;

        % check if next z-pos passes the target plane
        strike_indices = active_indices(gm_X(3,end,active_indices) > 0);
        if ~isempty(strike_indices)
            display([ 'Timestep ' num2str(tstep) ': ' num2str(length(strike_indices)) ' strikes.' ]);
            % interpolate absolute strike time?
            gm_result(:,1,strike_indices) = squeeze(gm_X(:,end,strike_indices));
            gm_result(:,2,strike_indices) = squeeze(gm_V(:,end,strike_indices));
            gm_result(:,3,strike_indices) = repmat([ tstep-1 ; tstep ; tstep*dt*t0 ],[1 length(strike_indices)]);
            active_indices = active_indices( ~ismember(active_indices, strike_indices) );
        end
    end

    t_codist_result = codistributor1d(3, codistributor1d.unsetPartition, [3, 3, N_part]);
    t_codist_saved = codistributor1d(3, codistributor1d.unsetPartition, [3, saved_steps, N_part]);

    % gather() to copy from GPU RAM to Main Memory
    r_divres = codistributed.build(gather(gm_result), t_codist_result, 'noCommunication');
    r_divsavX = codistributed.build(gather(gm_X), t_codist_saved, 'noCommunication');
    r_divsavV = codistributed.build(gather(gm_V), t_codist_saved, 'noCommunication');
```

```
    end % spmd

    % gather() again to recombine distributed arrays
    r_result = gather(r_divres);
    r_savX = gather(r_divsavX);
    r_savV = gather(r_divsavV);

    toc
    'Stop'

    % results to output variables
    dist = v_distrib;
    resXVT = r_result;
    savedX = r_savX;
    savedV = r_savV;

end

function n = bxcalc(x,z,L_z2)
    n = -x*z/L_z2;
end

function n = bycalc(y,z,L_z2)
    n = -y*z/L_z2;
end

function n = bzcalc(z,L_z2)
    n = (1+z^2/L_z2);
end

function n = threenorm(x,y,z)

    n = sqrt(x^2+y^2+z^2);

end

function n = tcalc(bx,by,bz,c)

    n = c*[ bx by bz ].';

end

function n = scalc(t)

    n = 2*t./(1 + t.^2);

```

```
end

function d = build_distrib(v0, x_range, y_range, z_range, v_range, t_dtheta, t_domega)
    % Build particle distribution

    % initial positions x y z
    % initial velocities v theta phi (magnitude, azimuth, elevation)
    %   mag 25:2000 eV, azi 0:pi, el 0:pi/2
    % input as [ x y z v theta phi ] columns in v_distrib_raw

    % range of thetas, discard first (pole) and last (plane)
    t_range = 0+t_dtheta:t_dtheta:pi/2-t_dtheta;

    angle_list = [ 0 0 ];
    for i=1:length(t_range)
        theta = t_range(i);
        for omega=0:2*pi/round(2*pi*sin(theta)*t_dtheta/t_domega):2*pi
            angle_list = [ angle_list ; theta omega ];
        end
    end

    v_distrib_raw = zeros(6,length(x_range) * length(y_range) * length(z_range) * length(v_range) * length(angle_list));
    vdr_ind = 1;
    for i=1:length(x_range)
        for j=1:length(y_range)
            for k=1:length(z_range)
                for l=1:length(angle_list)
                    for m=1:length(v_range)
                        v_distrib_raw(:,vdr_ind) = [ x_range(i) y_range(j) z_range(k) v_range(m) angle_list(l,1) angle_list(l,2) ];
                        vdr_ind = vdr_ind + 1;
                    end
                end
            end
        end
    end

    % Transform v_mag,theta,phi to v_x, v_y, v_z
    v_distrib = v_distrib_raw;
    % number is 2/(m_e*c^2) in eV^-1
    t_v = sqrt(3.913903e-6*v_distrib_raw(4,:))/v0;
    v_distrib(4,:) = t_v .* cos(v_distrib_raw(6,:)) .* cos(v_distrib_raw(5,:));
    v_distrib(5,:) = t_v .* cos(v_distrib_raw(6,:)) .*
```

```
        sin(v_distrib_raw(5,:));
    v_distrib(6,:) = t_v .* sin(v_distrib_raw(6,:));

    d = v_distrib;
end

function d = test_distrib()

    test_array = [  1 3 8 ;
                    4 7 2 ;
                    9 1 7 ;
                    4 6 2 ;
                    1 1 1 ];

    test_v = [ 0 2 1.22 ;
               0 1.9 1.22 ;
               0 2.1 1.22 ;
               0 2 1.12 ;
               0 2 1.32 ];

    d = shiftdim([ test_array test_v ],1);
end

function [ Xg, Yg, Zg, Bx, By, Bz ] = build_field()
    % Field Initialization
    L_xyz = 100;
    n_xyz = 100;
    k = pi/L_xyz;
    x0 = 51; y0 = 51; z0 = 51;

    % Generate the grid of the magnetic field
    [X, Y, Z] = meshgrid(1:100, 1:100, 1:1000);
    B_xgrid = zeros(100,100,1000); B_ygrid = zeros(100,100,1000); B_zgrid =
        zeros(100,100,1000);
    %B_maggrid = zeros(100,100,100);
    for ii = 1:100
        for jj = 1:100
            for kk = 1:1000
    %             B_xgrid(ii,jj,kk) = 0;
    %             B_ygrid(ii,jj,kk) = 0;
    %             B_zgrid(ii,jj,kk) = -1;
                % 2.85966 factor normalizes field so max magnitude is 1
                B_xgrid(ii,jj,kk) = -(5/8) * k * sin(k*(kk-z0)) * (ii - x0)
                    / 2.85966;
                B_ygrid(ii,jj,kk) = -(5/8) * k * sin(k*(kk-z0)) * (jj - y0)
                    / 2.85966;
                B_zgrid(ii,jj,kk) = 5 * (1 - .5*(1 + 1/8*k^2 * ((ii - x0)^2
```

```
                    + (jj - y0)^2)) * cos(k*(kk-z0))) / 2.85966;
                %   B_maggrid(ii,jj,kk) = sqrt(B_xgrid(ii,jj,kk)^2 +
                    B_ygrid(ii,jj,kk)^2 + B_zgrid(ii,jj,kk)^2);
            end
        end
    end

    Xg = X; Yg = Y; Zg = Z;
    Bx = B_xgrid; By = B_ygrid; Bz = B_zgrid;
end

function ok = selectGPUDeviceForLab()

    persistent hasGPU;

    if isempty( hasGPU )
        devIdx = mod(labindex-1,gpuDeviceCount())+1;
        try
            dev = gpuDevice( devIdx );
            hasGPU = dev.DeviceSupported;
        catch %#ok
            hasGPU = false;
        end
    end
    ok = hasGPU;

end
```

B.1.5 Support Scripts

This is a simple python script which takes the desired number of shards and cores per shard, as well as cell, wall time per core, and a run-identification number. It runs the Distribute function to create the master and shard-input files, and creates a PBS script using the template file (below), and accompanying submission script.

There is one problem with this, related to the fact that the script does not know how many total particles the Distribute function will be creating. If you're going for a one-particle-per-core scenario, and your particles are not evenly divided by your number of cores per shard, then you can end up with a scenario where on some shards you have more cores than particles, and end up with Matlab workers crashing and messing things up (and crashed workers don't give very nice feedback).

Fixing this requires knowing how Matlab's distributed() function split up the array. The script will try to figure that out by looking at input file sizes, but you can specify it manually with the `-m` option. Either way, we'll set up two PBS scripts, and a two-stage submission script to switch between the two. To disable this behavior (e.g. if not aiming for a 1:1 particle:core ratio) , use `-m -1`.

Finally, there's a simple script to run the Gather function.

`mss.py`:

```
#!/usr/bin/env python

from optparse import OptionParser
import subprocess
import sys

parser = OptionParser("Usage: %prog -r <run number> [options]")
parser.add_option("-r","--run",dest="run",type="int",default=0,
    help="Run number [required].")
parser.add_option("-c","--cell",dest="cell",type="string",default="j",
    help="Cell to use [default: %default].")
parser.add_option("-s","--shards",dest="shards",type="int",default=20,
    help="Number of shards to break distribution into [%default].")
parser.add_option("-n","--cores",dest="cores",type="int",default=24,
    help="Number of cores per shard [%default].")
parser.add_option("-w","--wall",dest="wall",type="int",default=50,
    help="Wall time per core [%default].")
parser.add_option("-m","--modulo",dest="modulo",type="int",default=0,
    help="Manually specify modulo point.  The first M shards will be given N cores per shard, the remainder will be given N-1.  Set to -1 to disable auto-detect.")

(opt, args) = parser.parse_args()

if opt.run==0:
    print("Must provide a run number with -r.")
    sys.exit()

# break up distribution
subprocess.call(["matlab","-nodisplay","-r",
    "try; mirror_shards_distribute({0},{1}); catch; end; quit".format(opt.run,opt.shards)])

def ms_size(i,t):
    # returns size of an mshard file
    fname = "mshard-r4-{0}of{1}-input.mat".format(i,t)
    return os.path.getsize(fname)

if opt.modulo==0:
    # try to figure out where Distribute put the modulo point
    size = ms_size(1,opt.shards)
    for i in range(2,opt.shards+1):
```

```
        if size != ms_size(i,opt.shards):
            opt.modulo = i-1
            break
        if i==opt.shards:
            opt.modulo = opt.shards

elif opt.modulo == -1:
    # auto-detect disabled
    opt.modulo = opt.shards

def mss_files(cores,wall,cell,run,mtag):
    # function to create the PBS scripts
    pbsfn = "mss_PBS-r{0}-m{1}.sh".format(run,mtag)
    pbsfile = open(pbsfn,"w")

    subprocess.call(["sed",
        "s/@@PPN@@/{0}/; s/@@WALL@@/{1}/; s/@@CELL@@/cell{2}/; s/@@RUN@@/{3}/;".format(cores,wall,cell,run),
            "./mirror_shards.PBStemplate"],
            stdout=pbsfile)
    pbsfile.close()

# create primary PBS and job submission scripts
mss_files(opt.cores,opt.wall,opt.cell,opt.run,0)

subfile = open("mss_submit-r{0}.sh".format(opt.run),"w")
subfile.write("#!/bin/bash\n\n")
subfile.write("for ((i=1 ; i<={0} ; i++)); do\n".format(opt.modulo))
subfile.write("\tqsub -t $i mss_PBS-r{0}-m{1}.sh\n".format(opt.run,0))
subfile.write("\tsleep 7\n")
subfile.write("done\n")

if opt.modulo != opt.shards:
    # if modulo, create second PBS script,
    # add second part to submission script
    mss_files(opt.cores-1,opt.wall,opt.cell,opt.run,1)

    subfile.write("\nfor ((i={0} ; i<={1} ; i++)); do\n".format(opt.modulo+1,opt.shards))
    subfile.write("\tqsub -t $i mss_PBS-r{0}-m{1}.sh\n".format(opt.run,1))
    subfile.write("\tsleep 7\n")
    subfile.write("done\n")

subfile.close()

# clean up output from Distribute
subprocess.call("rm -rf Job1*",shell=True)
```

mirror_shards.PBStemplate:

```
#!/bin/bash -l
# declare a name for this job to be sample_job
#PBS -N mirror_shard

# request the default queue for this job
#PBS -q default

#PBS -l nodes=1:ppn=@@PPN@@
#PBS -l walltime=@@WALL@@:00:00
#PBS -l feature='@@CELL@@'

# mail is sent to you when the job starts
# and when it terminates or aborts
#PBS -m bea

# specify your email address
#PBS -M micah.p.dombrowski.gr@dartmouth.edu

#change to the directory where you submitted the job
cd $PBS_O_WORKDIR

# include the relative path to the name of your MPI program
matlab -nodisplay -r "try; mirror_shards_alice($PBS_ARRAYID,@@PPN@@,
    'mshards-r@@RUN@@-master.mat'); catch; end; quit"
```

msf.py:

```
#!/usr/bin/env python

from optparse import OptionParser
import subprocess, sys, os

parser = OptionParser("Usage: %prog -r <run number> [options]")
parser.add_option("-r","--run",dest="run",type="int",default=0,
    help="Run number [required].")

(opt, args) = parser.parse_args()

if opt.run==0:
    print("Must provide a run number with -r.")
    sys.exit()

# gather distribution
subprocess.call(["matlab","-nodisplay","-r",
    "try; mirror_shards_gather('mshards-r{0}-master.mat'); catch; end;
        quit".format(opt.run)])
```

B.2 Result Reformation

These functions and codes take the raw output from the Mirror Shards system, and turn it into something neatly packaged and usable in the later stages. The data Mirror Shards returns is: three matrices, of the final 1,000 points for each particle, of position, velocity, and magnetic field, and a 'result' array of data regarding travel times.

B.2.1 Gyro-Interpolation

Takes the raw data for each particle (the last 1,000 positions and velocities) and fits an interpolating gyro-orbit function to it, then uses that to interpolate to the actual strike values at the target plane. Uses the `HyperSVD()` algebraic circle-fitting function by Nikolai Chernov (http://people.cas.uab.edu/~mosya/cl/), included below.

gyroterpolate.m

```
function [ t_tz_time, t_Xf, t_Vf, t_mphi, t_circ ] = gyroterpolate(t_X, t_V, t_B, target_z, dt, t_d)
% Takes 3xTxN input vectors, where T = some number of saved timesteps,
% and N = some number of particles which have crossed the target
% z-level.  Fits a gyro-orbit to each particle's track, then
% interpolates the actual strike XVT.

    Ns = size(t_X, 2);
    Np = size(t_X, 3);
    t_tz_time = zeros(Np,1);
    t_Xf = zeros(Np,3);
    t_Vf = zeros(Np,3);
    t_mphi = zeros(Np,1);
    t_circ = zeros(Np,3);

    options = optimoptions('fminimax');
    options.Display = 'none';

    parfor part=t_d%1:Np

        t_pX = squeeze(t_X(:,:,part));
        t_pV = squeeze(t_V(:,:,part));

        tz_center = HyperSVD(squeeze(t_pX(1:2,:)).');

        if tz_center(3) ~= 0 % gyropath
```

```
        t_circ(part,:) = tz_center;

        % We know the equations of motion that the particle should
        % be following; the only thing we don't know is the phase.
        omega = sqrt(sum(squeeze(t_B(:,:,part)).^2,1)); % Bmag
        vperp = sqrt( squeeze(t_pV(1,:)).^2 + squeeze(t_pV(2,:)).^2 );
        vpar = t_pV(3,:);
        ttime = (-(Ns-2):1)*dt;

        t_Xc = [ t_pX(1,:) ; t_pX(2,:) ; t_pX(3,:) ];
        t_x0 = t_pX(1,end-1);
        t_y0 = t_pX(2,end-1);
        t_z0 = t_pX(3,end-1);
        deltx = t_Xc(1,:);
        delty = t_Xc(2,:);
        deltz = t_Xc(3,:)-t_z0;
        Cx = -vperp./omega;
        Cy =  vperp./omega;
        Cz = vpar.*ttime;
        tau = omega.*ttime;

        % geometric error
        ferrorphi = @(phi) ferrorphifunc(phi, deltx, delty, deltz, tau, Cx, Cy, Cz);

        [ t_mphi(part), ~ ] = fminimax(ferrorphi, pi, [], [], [], [], 0, 2*pi, [], options);

        % X, V before target crossing
        tz_x0 = t_pX(1,end-1); tz_vx0 = t_pV(1,end-1);
        tz_y0 = t_pX(2,end-1); tz_vy0 = t_pV(2,end-1);
        tz_z0 = t_pX(3,end-1); tz_vz0 = t_pV(3,end-1);
        % travel time to target crossing
        t_tz_time(part) = (target_z-tz_z0)/vpar(end-1);

        % now just use gryo equations to get final interp. results
        tz_time = t_tz_time(part);
        tz_vperp = sqrt(tz_vx0^2 + tz_vy0^2);
        tz_omega = omega(end-1);
        tz_tau = tz_omega*(ttime(end-1)+tz_time);
        tz_phi = t_mphi(part);

        tz_xf = -tz_vperp/tz_omega*cos(tz_tau + tz_phi) + tz_center(1);
        tz_yf =  tz_vperp/tz_omega*sin(tz_tau + tz_phi) + tz_center(2);
        tz_zf =  tz_vz0*tz_time + tz_z0;

```

```
            tz_vxf = tz_vperp*sin(tz_tau + tz_phi);
            tz_vyf = tz_vperp*cos(tz_tau + tz_phi);
            tz_vzf = tz_vz0;

            [ tz_x0 tz_y0 tz_z0 ];
            t_Xf(part,:) = [ tz_xf tz_yf tz_zf ];
            [ tz_vx0 tz_vy0 tz_vz0 ];
            t_Vf(part,:) = [ tz_vxf tz_vyf tz_vzf ];

        else % straight line

            % travel time to target crossing
            t_tz_time(part) = (target_z-t_pX(3,end-1))/t_pV(3,end-1);

            % x, y, and velocities don't change, just set z = target_z
            t_Xf(part,:) = [ t_pX(1,end-1) t_pX(2,end-1) target_z ].';
            t_Vf(part,:) = t_pV(:,end-1).';
            t_mphi(part) = NaN;
            t_circ(part,:) = [ t_pX(1,end-1) t_pX(2,end-1) 0 ];

        end



    end

    %fdx = fdx + t_circ(1);
    %fdy = fdy + t_circ(2);

    %    phi = fminbnd(ferrorphi, 0, 2*pi);

end

function err = ferrorphifunc(phi,deltx,delty,deltz,tau,Cx,Cy,Cz)

    err = sqrt( ...
        (deltx - Cx.*cos(tau + phi)).^2 + ...
        (delty - Cy.*sin(tau + phi)).^2 + ...
        (deltz - Cz).^2);

end

function vp = vel_upd(t, v, B, phi)
    v_perp = sqrt(v(1).^2 + v(2).^2);
    omega_g = 2*pi*B;

    vp(1) = v_perp.*sin(omega_g.*t + phi);
```

```
    vp(2) = v_perp.*sin(omega_g.*t + phi);
    vp(3) = v(3);
end

function xp = pos_upd(t, x, v, B, phi)
    v_perp = sqrt(v(1).^2 + v(2).^2);
    omega_g = 2*pi*B;

    xp(1) = x(1) + -v_perp/omega_g.*cos(omega_g.*t + phi);
    xp(2) = x(2) +  v_perp/omega_g.*sin(omega_g.*t + phi);
    xp(3) = x(3) + v(3).*t;
end
```

HyperSVD.m

```
function Par = HyperSVD(XY)
%--------------------------------------------------------------------
%
%     Algebraic circle fit with "hyperaccuracy"
%     (with zero essential bias)
%
%     Input:  XY(n,2) is the array of coordinates
%             of n points x(i)=XY(i,1), y(i)=XY(i,2)
%
%     Output: Par = [a b R] is the fitting circle:
%             center (a,b) and radius R
%
%     Note: this is a version optimized for stability, not for speed
%
%--------------------------------------------------------------------

centroid = mean(XY);   % the centroid of the data set

X = XY(:,1) - centroid(1);  %  centering data
Y = XY(:,2) - centroid(2);  %  centering data
Z = X.*X + Y.*Y;
ZXY1 = [Z X Y ones(length(Z),1)];
[U,S,V]=svd(ZXY1,0);
if (S(4,4)/S(1,1) < 1e-12)   %  singular case
    A = V(:,4);
else                         %  regular case
    R = mean(ZXY1);
    N = [8*R(1) 4*R(2) 4*R(3) 2; 4*R(2) 1 0 0; 4*R(3) 0 1 0; 2 0 0 0];
    W = V*S*V';
    [E,D] = eig(W*inv(N)*W);
    [Dsort,ID] = sort(diag(D));
    Astar = E(:,ID(2));
```

```
    A = W\Astar;
end

Par = [-(A(2:3))'/A(1)/2+centroid ,
    sqrt(A(2)*A(2)+A(3)*A(3)-4*A(1)*A(4))/abs(A(1))/2];

end   %  HyperSVD
```

B.2.2 Hemispherical Filling

Takes particles which were launched at one azimuthal angle, and rotates the whole system to get a gyrotropic set with a constant solid angle subtended.

hemi_fill.m

```
function [ n, n_azi, rel ] = hemi_fill(xvt,r_dist,t_dphi,t_domega,varargin)
% hemi_fill: function to populate a constant-solid-angle hemisphere,
% given a single stripe of co-latitude positions and velocities, the
% corresponding co-latitudes, and the solid angle value in steradians.
%
% xv should be a 3x2xN vector, where N=(2pi/dphi)-2,
% positions are in (:,1,:), and velocities in (:,2,:)

    opt = struct('cell',false,'phistop',2*pi);
    opt = optParse(opt,varargin{:});

    % range of thetas, discard first (pole) and last (equator)
    t_phis = 0+t_dphi:t_dphi:pi/2-t_dphi;
    n_phis = length(t_phis);
    t_alpha = sqrt(r_dist(4,:).^2 + r_dist(5,:).^2)./r_dist(6,:);
    n_En = length(find(t_alpha == 0));

    nc_Xt = cell(n_phis,n_En,1);
    nc_Vt = cell(n_phis,n_En,1);
    nc_Xb = cell(n_phis,n_En,1);
    nc_Vb = cell(n_phis,n_En,1);
    nc_t = cell(n_phis,n_En,1);
    n_azi = zeros(n_phis,1);
    nc_rel = cell(n_phis,n_En,1);
    for i=1:n_phis
        t_phi = t_phis(i);

        n_azi(i) = round(2*pi*sin(t_phi)*t_dphi/t_domega);
        l_thetas = 0:2*pi/n_azi(i):opt.phistop;
        n_az = length(l_thetas);
        [ n_az n_azi(i) ];
```

```
        if n_az ~= n_azi(i)
        %    display('fuuu');
            n_azi(i) = n_az;
        end
        for j=1:n_En
            part = (i-1)*n_En + j;
            display(['fnh ' num2str(part) ' lsjdf ' num2str(size(xvt))])
            t_x = squeeze(xvt(:,1,part));
            t_v = squeeze(xvt(:,2,part));
            b_x = r_dist(1:3,part);
            b_v = r_dist(4:6,part);
            t_t = squeeze(xvt(:,3,part));

            n_xt = zeros(3,n_az);
            n_vt = zeros(3,n_az);
            n_xb = zeros(3,n_az);
            n_vb = zeros(3,n_az);
            for k=1:n_az

                t_theta = l_thetas(k);
                t_rot = [ cos(t_theta) sin(t_theta) 0 ; -sin(t_theta)
                    cos(t_theta) 0 ; 0 0 1 ];

                n_xt(:,k) = t_rot*t_x;
                n_vt(:,k) = t_rot*t_v;
                n_xb(:,k) = t_rot*b_x;
                n_vb(:,k) = t_rot*b_v;
            end

            nc_Xt{i,j} = n_xt;
            nc_Vt{i,j} = n_vt;
            nc_Xb{i,j} = n_xb;
            nc_Vb{i,j} = n_vb;
            nc_t{i,j} = t_t(3);
            nc_rel{i,j} = part;
        end

    end

    n_vec = sum(n_azi)*n_En;
    if opt.cell
        n = cell(n_phis,5);
        n(:,1) = nc_Xt;
        n(:,2) = nc_Vt;
        n(:,3) = nc_Xb;
        n(:,4) = nc_Vb;
        n(:,5) = nc_t;
```

```
    else
        n = zeros(13,n_vec);
        rel = zeros(1,n_vec);
        i_n = 0;
        for i=1:n_phis
            for j=1:n_En
                for k=1:n_azi(i)
                    i_n = i_n + 1;
                        n(:,i_n) = [ nc_Xt{i,j}(:,k) ; nc_Vt{i,j}(:,k) ;
                            nc_Xb{i,j}(:,k) ; nc_Vb{i,j}(:,k) ; nc_t{i,j} ];
                    rel(i_n) = nc_rel{i,j};
                end
            end
        end
    end

end

function optstr = optParse(options, varargin)

    %# read the acceptable names
    optionNames = fieldnames(options);

    %# count arguments
    nArgs = length(varargin);
    if round(nArgs/2)~=nArgs/2
       error('hemi_fill needs propertyName/propertyValue pairs')
    end

    for pair = reshape(varargin,2,[]) %# pair is {propName;propValue}
       inpName = lower(pair{1}); %# make case insensitive

       if any(strcmp(inpName,optionNames))
          %# Overwrite options. If you want you can test for the right
          %# class here.  Also, if you find out that there is an option
          %# you keep getting wrong, you can use "if strcmp(inpName,
          %# 'problemOption'),testMore,end"-statements
          options.(inpName) = pair{2};
       else
          error('%s is not a recognized parameter name',inpName)
       end
    end

    optstr = options;
end
```

B.2.3 Data-Processing Utility Script

This script runs `gryoterpolate()` and `hemi_fill()`, fiddling with the data in between and after to yield a nicely structured result matrix for use in distribution building.

```
%% Mirror data raw output manipulation

% Important constants
Np = size(r_savX, 3);
B0 = 50e-6; % Magnetic field base is 50 uT
v0 = 0.00989179273; % likewise velocity base in
                    % PSL is equivalent to 25 eV
r0 = 0.337212985; % based on Larmour radius w/ above,
                  % length base is ~0.337 m
t0 = 7.14477319e-7; % based on B, Larmour period ~714 ns in s
dt = 0.01;

%% Gyro-interpolation

[ t_tz_time, t_Xf, t_Vf, t_mphi, t_circ ] = gyroterpolate(r_savX,r_savV,r_savB,0,dt,1:Np);

%% Build new results matrix

r_interp = zeros(3, 3, Np);

% new layout:
% 1,2 position and velocity at target-z
% 3,  number of timesteps before target (unitless, # of timesteps),
%     fractional timestep to actually reach target (unitless, simulation time [timesteps*dt]),
%     total time (in ns, timesteps*dt*t0)

r_interp(:,1,:) = t_Xf.';
r_interp(:,2,:) = t_Vf.';

t_base_time = squeeze(r_res(2,3,:))-1;
t_time_ns = (t_base_time*dt+t_tz_time)*t0;

r_interp(:,3,:) = [ squeeze(r_res(2,3,:))-1 t_tz_time t_time_ns ].';

%% Hemi_fill to build a gyrotropic distribution.

t_dphi = 3*pi/256; % delta for co-latitude
t_domega = 0.001; % delta for solid angle in steradians
t_phis = 0+t_dphi:t_dphi:pi/2-t_dphi; % range of phis, discard
                                      % first (pole) and last (plane)

```

```
% final data is stored flat in a 13xN matrix
% top x y z top vx vy vz bottom x y z bottom vx vy vz time (ns)

[ r_hemiterp, ~, r_hemirel ] = hemi_fill(r_interp, r_dist, t_dphi, t_domega);

%% Save intermediate data

save J:\Data' Core'\particles\sim\run4-hemi.mat t_domega t_dphi t_phis B0 r0
    t0 v0 target_length target_z N_part dt mirror_ratio length_factor r_dist
    r_res r_interp r_hemiterp

%% Convert to units and reverse all velocities

Np = size(r_hemiterp,2);
eVconst = 3.913903e-6;

t_X_t = r_hemiterp(1:3,:)*r0; % Distances in m

t_vx_t = -r_hemiterp(4,:);
t_vy_t = -r_hemiterp(5,:);
t_vz_t = -r_hemiterp(6,:);
t_vmag_t = sqrt(t_vx_t.^2 + t_vy_t.^2 + t_vz_t.^2); % v, unitless
t_vpar_t = t_vz_t; % vpar, unitless
t_vper_t = sqrt(t_vx_t.^2 + t_vy_t.^2); % vper, unitless

t_alpha_t = atan2(t_vper_t,-t_vpar_t);%*180/pi;
t_theta_t = 2*pi-atan2(t_vy_t,t_vx_t); % math to convert from atan2 output
    range to standard
t_theta_t(t_theta_t>=2*pi) = t_theta_t(t_theta_t>=2*pi)-2*pi; % 0to2pi
    clockwise from +x angle
%t_theta_t = t_theta_t * 180/pi;

t_X_b = r_hemiterp(7:9,:)*r0; % Distances in m

t_En_t = (t_vmag_t*v0).^2/eVconst; % Energy, eV
t_vmag_t_mps = t_vmag_t*v0*299792458; % convert to m/s
t_vpar_t_mps = t_vpar_t*v0*299792458;
t_vper_t_mps = t_vper_t*v0*299792458;

t_vx_b = -r_hemiterp(10,:);
t_vy_b = -r_hemiterp(11,:);
t_vz_b = -r_hemiterp(12,:);
t_vmag_b = sqrt(t_vx_b.^2 + t_vy_b.^2 + t_vz_b.^2); % v, unitless
t_vpar_b = t_vz_b; % vpar, unitless
t_vper_b = sqrt(t_vx_b.^2 + t_vy_b.^2); % vper, unitless

t_alpha_b = atan2(t_vper_b,-t_vpar_b);%*180/pi;
```

```
t_theta_b = 2*pi-atan2(t_vy_b,t_vx_b); % math to convert from atan2 output
    range to standard
t_theta_b(t_theta_b>=2*pi) = t_theta_b(t_theta_b>=2*pi)-2*pi; % 0to2pi
    clockwise from +x angle
%t_theta_b = t_theta_b * 180/pi;

t_En_b = (t_vmag_t*v0).^2/eVconst; % Energy, eV
t_vmag_b_mps = t_vmag_b*v0*299792458; % convert to m/s
t_vpar_b_mps = t_vpar_b*v0*299792458;
t_vper_b_mps = t_vper_b*v0*299792458;

% 17xN full-results matrix
% (x,y,z, vmag, vpar, vper, pa, azi)_top
% (x,y,z, vmag, vpar, vper, pa, azi)_bottom
% time

%r_mirror_XVPA = [ ...
%    t_X_t(1,:) ; t_X_t(2,:) ; t_X_t(3,:) ; ...
%    t_vmag_t_mps ; t_vpar_t_mps ; t_vper_t_mps ; ...
%    t_alpha_t ; t_theta_t ; ...
%    t_X_b(1,:) ; t_X_b(2,:) ; t_X_b(3,:) ; ...
%    t_vmag_b_mps ; t_vpar_b_mps ; t_vper_b_mps ; ...
%    t_alpha_b ; t_theta_b ; ...
%    r_hemiterp(13,:) ...
%    ];

% Discard X (assume homogeneity), re-add Energies
% 13xN matrix
% (En, vmag, vpar, vper, pa, azi)_top
% (En, vmag, vpar, vper, pa, azi)_bottom
% time

r_mirror_EVPA = [ ...
    t_En_t ; ...
    t_vmag_t_mps ; t_vpar_t_mps ; t_vper_t_mps ; ...
    t_alpha_t ; t_theta_t ; ...
    t_En_b ; ...
    t_vmag_b_mps ; t_vpar_b_mps ; t_vper_b_mps ; ...
    t_alpha_b ; t_theta_b ; ...
    r_hemiterp(13,:) ...
    ];

% Build a map structure, to keep track of what's what.

smap_EVPA.top.En = 1;
smap_EVPA.top.v.mag = 2;
smap_EVPA.top.v.para = 3;
```

```
smap_EVPA.top.v.perp = 4;
smap_EVPA.top.alpha = 5;
smap_EVPA.top.theta = 6;
smap_EVPA.bot.En = 7;
smap_EVPA.bot.v.mag = 8;
smap_EVPA.bot.v.para = 9;
smap_EVPA.bot.v.perp = 10;
smap_EVPA.bot.alpha = 11;
smap_EVPA.bot.theta = 12;
smap_EVPA.time = 13;
```

B.3 Distribution Building and Reduction, Growth Rates

B.3.1 Maxwell-Boltzmann Distribution

Takes parameters, and arrays that tell it where its sample points in energy pitch-angle phase space are, and builds a Maxwellian distribution.

maxwellian.m

```
function [ maxw_vel ] = maxwellian( temp, shift_eV, PAcenter, PAwidth, in_velocities, in_angles)
%maxwellian(temp, PA center, PA width, input eneriges, input angles)
% Returns a discretely sampled, joint probability distribution
% function, based on the input parameters, sampled at the provided
% energies and angles.  Temp in K, shift in eV, PA in degrees, leave
% PAcenter empty [] to use a flat pitch-angle distribution.

    v_th = sqrt(3*temp*15156333.1);     % Convert input temp. to
                                        % v_th = (3kT/m)^(1/2)
    eVconst = 3.913903e-6; % 2/(m_e*c^2) in eV^-1,
                           % i.e. conversion from eV to PSL
    v0 = 0.00989179273; % velocity base in PSL is equivalent to 25 eV

    shift = sqrt(shift_eV*eVconst)*299792458; % conv shift eV to m/s

    % Maxwell-Boltzmann in velocity
    % maxw_vel = (temp/pi)^(3/2) * 4*pi * (in_velocities).^2 .* exp(-temp*(in_velocities-shift).^2);
    maxw_exp = exp(-(in_velocities-shift).^2/(2*v_th^2));
    maxw_vel = (2*pi)^(-3/2)*v_th^-3 .* maxw_exp;

    if ~isempty(PAcenter)
```

```
        % Gaussian in pitch angle
        maxw_PA = 1/(PAcenter*sqrt(2*pi)) *
            exp(-(in_angles-PAwidth).^2/(2*PAcenter^2));
        maxw_vel = maxw_vel.*maxw_PA;
    end

end
```

B.3.2 Background/Beam Definition Structure Builder

This takes a launch period, sampling period, and structures that define the ionospheric background, secondary background, and beams, and builds a parent structure encompassing all of that. It performs a couple of simple sanity checks, and builds a vector with the start times for each segment of the beam structure, as well as the 'end' time.

`build_dyn_struct()`

```
function s_dyn = build_dyn_struct( launch_dt, sample_dt, s_iono, s_bg,
    s_beams )
%build_dyn_struct Builds a dynamic distribution definition structure.
% Builds a structure for feeding to dynamic_distribution().
% Beyond using struct(), the main function is to build the vector
% s_dyn.times, which has the start time of each beam, so you can do a
% simple find(s_dyn.times >= time & s_dyn.times < time) to figure out
% what beam def is active at a given time.

n_beams = length(s_beams);
v_times = zeros(n_beams+1,1);
for i=2:n_beams+1
    v_times(i) = v_times(i-1) + s_beams{i-1}.dwell_time;
end

if sample_dt < 10*launch_dt
    display('Sampling time should really be at least 10 times launch time!')
end

if ~isempty(find(diff(v_times) < launch_dt, 1))
    display('Dwell time lower than launch time!  Is this really what you
        want?')
end

s_dyn = struct('launch_dt', launch_dt, 'sample_dt', sample_dt, ...
    'iono', s_iono, 'bg', s_bg, 'times', v_times);
s_dyn.beams = s_beams;

end
```

B.3.3 Dynamic Distribution Timeslice Calculator

Takes the a time, the input data from the test particle simulation and its map structure, and a definition structure made by `build_dyn_struct()`, and returns the Maxwellian for the given time.

`dynamic_distribution.m`

```
function [ dist, sdist, n_beam ] = dynamic_distribution(time, in_dist,
   in_map, dist_def)
% Returns a distribution at a given time, for a provided 2xN list of
% particles/distribution function element centers, with velocities
% in (1,:) and pitch angles in (2,:).  Returns an N-element list which
% is values of f(vmag,pa) for each particle.

    in_vmag = in_dist(in_map.bot.v.mag,:);
    in_PA = in_dist(in_map.bot.alpha,:);

    % ionosphere parameters
    iono_def = dist_def.iono;

    % Create ionospheric distribution
    iono_dist = maxwellian(iono_def.temp, iono_def.shift, ...
        iono_def.PAcenter, iono_def.PAwidth, in_vmag, in_PA);
    iono_part = iono_dist*iono_def.n;

    % background parameters
    bg_def = dist_def.bg;

    % Create background distribution
    bg_dist = maxwellian(bg_def.temp, bg_def.shift, ...
        bg_def.PAcenter, bg_def.PAwidth, in_vmag, in_PA);
    bg_part = bg_dist*bg_def.n;

    % We'll be using segment time /. dwell_time
    i_beam = find(dist_def.times <= time, 1, 'last');
    beam_def = dist_def.beams{i_beam};
    n_beam = beam_def.n;

    if beam_def.n == 0 % BG-only case

        dist = bg_part+iono_part;
        sdist = { iono_part, bg_part, zeros(size(iono_part)) };

    else % BG + beam

        beam_dist = maxwellian(beam_def.temp, beam_def.shift, ...
            beam_def.PAcenter, beam_def.PAwidth, in_vmag, in_PA);
```

```
        beam_part = beam_dist*beam_def.n;

        dist = iono_part+bg_part+beam_part;
        sdist = { iono_part, bg_part, beam_part };

    end

end
```

B.3.4 Azimuthal Summation

The first step of reduction is to undo all that hard work `hemi_fill()` did. In gyrotropic cases the azimuthal angle has no effect on time of flight, so we can do this before we deal with any time summation issues.

This function will be run many times, and the `uniquetol()` required to get the indices of unique $v_\perp$ and $v_\parallel$ is quite slow. The result is also identical for a given set of input velocity vectors, so we can save the result, and reuse it for every `azi_sum()` in a given run. This is what `azi_sum_stash()`, with its friend `array_checksum()`, both included below, accomplish: compare input vs. stored checksums, and if it checks out, just pass back saved results. Saves and checksums are stored as persistent variables in the function-local context.

Finally, also included is a simple intermediate utility function `time_azi_sum_chain()`, which chains `dynamic_distribution()` to `azi_sum()`, for great justice.

azi_sum.m

```
function [ m_fperppara, out_map, m_intersect, s_uniques ] = azi_sum(in_dist,
    in_map, t_dalpha, t_domega)
% Takes a 14xN list of cells in a distribution in 3-D perp/para/azi
% space, and sums over Azimuthseseses to return a 12xM list of 2-D
% reduced distribution functions in v_perp-v_para space.  Optionally
% returns its intersection list and a structure of the unique perp
% and para values.

    % Input 14xN matrix
    % (En, vmag, vpar, vper, pa, azi)_top
    % (En, vmag, vpar, vper, pa, azi)_bottom
    % time, distN

    % Returns 12xM
    % (En, vmag, vpar, vper, pa)_top
    % (En, vmag, vpar, vper, pa)_bottom
    % time, N

    v_perp = in_dist(in_map.bot.v.perp,:);
```

```
    v_para = in_dist(in_map.bot.v.para,:);

    % get cell of tuple-matches
    [ m_intersect, s_uniques ] = azi_sum_stash(v_perp, v_para);
    n_cells = length(m_intersect);

    m_fperppara = zeros(12,n_cells);
    for i=1:n_cells
        v_indices = m_intersect{i};

        v_distN = in_dist(in_map.dist,v_indices);

        % Since these are limited to a single vperp,vpara, they all
        % have the same alpha, i.e. they're in an azimuthal ring.
        % Because that's exactly how azimuths were defined.  Thus,
        % dtheta is just 2pi/(# of points).  We can just factor that.
        t_dtheta = 2*pi/length(v_indices);
        t_sumN = sum(v_distN*t_dtheta);

        m_fperppara(:,i) = [ in_dist([ ...
            in_map.top.En in_map.top.v.mag ...
            in_map.top.v.perp in_map.top.v.para in_map.top.alpha ...
            in_map.bot.En in_map.bot.v.mag ...
            in_map.bot.v.perp in_map.bot.v.para in_map.bot.alpha ...
            in_map.time],v_indices(1)) ; t_sumN ];
            % The values from the input should be
            % identical for all v_indices()

    end

    % create new output field map
    out_map.top.En = 1; out_map.top.v.mag = 2;
    out_map.top.v.perp = 3; out_map.top.v.para = 4;
    out_map.top.alpha = 5;
    out_map.bot.En = 6; out_map.bot.v.mag = 7;
    out_map.bot.v.perp = 8; out_map.bot.v.para = 9;
    out_map.bot.alpha = 10;
    out_map.time = 11; out_map.dist = 12;

end
```

azi_sum_stash.m

```
function [ m_intersect, s_uniques ] = azi_sum_stash(v_perp,v_para)
% Finds unique (v_perp,v_para) tuples and returns the indices from the
% data that hit those tuples, i.e. a cell of arrays of data points with
% the same (v_perp,v_para), but different azimuths.
```

```
% Will cache this search for inputs which match checksums, because the
% uniquetol()s and intersections are rather slow.

    % uniquetol() and the set intersection stuff are very time
    % consuming, so we'll cache those results and a checksum.
    persistent perpcs paracs sm_intersect ss_uniques

    % First check if we've got an accurate cache.
    newperpcs = array_checksum(v_perp); % checksum of perp velocities
    newparacs = array_checksum(v_para); % checksum of para velocities

    if ~isequal(perpcs,newperpcs) || ~isequal(paracs,newparacs) ||
        isempty(sm_intersect)
        display('Rerunning uniquetol() & intersections.')
        % no stored copy or checksums were bad, must run uniquetol()s

        perpcs = newperpcs; % store checksums
        paracs = newparacs;

        [ v_vperpvals, v_vperpinds ] =
            uniquetol(v_perp,0.000001,'OutputAllIndices',true);
        [ v_vparavals, v_vparainds ] =
            uniquetol(v_para,0.000001,'OutputAllIndices',true);

        ss_uniques.v_vperpvals = v_vperpvals; ss_uniques.v_vperpinds =
            v_vperpinds;
        ss_uniques.v_vparavals = v_vparavals; ss_uniques.v_vparainds =
            v_vparainds;

        n_vperp = length(v_vperpvals);
        n_vpara = length(v_vparavals);

        % Make a grid for all possible (v_perp,v_para) tuples
        m_intersect = cell(n_vperp,n_vpara);
        m_interlen = zeros(n_vperp,n_vpara);
        for i=1:n_vperp
            parfor j=1:n_vpara

                % v_indices = intersect(v_vperpinds{i},v_vparainds{j});
                % using ismember() is faster, but still pretty slow
                m_intersect{i,j} = v_vperpinds{i}(ismember(v_vperpinds{i},
                    v_vparainds{j}));
                m_interlen(i,j) = length(m_intersect{i,j});

            end
        end

```

```
        % flatten
        m_intersect = reshape(m_intersect,1,[]);
        m_interlen = reshape(m_interlen,1,[]);

        % keep only points with matching cells
        m_intersect = m_intersect(m_interlen ~= 0);

        sm_intersect = m_intersect;
    end

    s_uniques = ss_uniques;
    m_intersect = sm_intersect;

end
```

array_checksum.m

```
function cs = array_checksum(in)

    flatsum = sum(in);
    cs = flatsum/sum((in/flatsum).^2);

end
```

time_azi_sum_chain.m

```
function [ m_EVPN, smap_EVPN ] = time_azi_sum_chain(in_time, in_dist,
    in_map, dist_def)

    t_dphi = 3*pi/256; % delta for co-latitude
    t_domega = 0.001; % delta for solid angle in steradians

    % generate dist at top-time t
    t_dist = dynamic_distribution(in_time, in_dist, in_map, dist_def);
    m_EVPAN = [ in_dist ; t_dist ]; % Tack distribution on to the rest
    smap_EVPAN = in_map;
    smap_EVPAN.dist = 14;

    % azi_sum
    [ m_EVPN, smap_EVPN ] = azi_sum(m_EVPAN, smap_EVPAN, t_dphi, t_domega);

end
```

B.3.5 Perpendicular Summation

Now we sum over the perpendicular velocities, to get a parallel reduced distribution function. This just straight up returns the RDF, no more structy stuff since we're combining things. I wonder how setting the bin centers arbitrarily might change things in the results...

perp_sum.m

```
function [ m_rdf, paravals ] = perp_sum(in_dist, in_map, paravals)
% Takes a 12xN distribution of cells in 2-D perp/para space,
% and sums over perp values to return a 10xM 1-D distribution.

    % Input 12xM
    % (En, vmag, vpar, vper, pa)_top
    % (En, vmag, vpar, vper, pa)_bottom
    % time, N

    v_perp = in_dist(in_map.bot.v.perp,:);
    v_para = in_dist(in_map.bot.v.para,:);
    v_N = in_dist(in_map.dist,:);

    n_para = length(paravals);

    [ para_widths, para_deltas ] = half_deltas(paravals);

    m_rdf = zeros(n_para,1);
    parfor i=1:n_para
        para = paravals(i);
        deltas = para_deltas(i:i+1);

        parainds = find(v_para >= para-deltas(1) & v_para < para+deltas(2));
        [ t_perpvals, t_perpinds ] = uniquetol(v_perp(parainds), ...
            0.00001, 'OutputAllIndices', true);
        n_perp = length(t_perpvals);

        if n_perp > 1
            % flatten the lists, making a list of all vals,
            % and a list of indices
            perpvals = [];
            for k=1:n_perp
                perpvals = [ perpvals
                    repmat(t_perpvals(k),1,length(t_perpinds{k})) ];
            end
            perpinds = vertcat(t_perpinds{:});

            [ s_perpvals, si_perpvals ] = sort(perpvals);

            % indices within this batch of parainds
```

```
            si_perpinds = perpinds(si_perpvals);
            % values of f(perp,para)
            s_distN = v_N(parainds(si_perpinds));

            % trapezoidal rule function,
            % 1/2 sum( (v_{i+1}-v_i)*(f(v_{i+1})+f(v_i))*v_i )
            delta_v = diff(s_perpvals);
            f_sums = s_distN(1:end-1) + s_distN(2:end);
            f = 0.5*sum( delta_v .* f_sums .* s_perpvals(1:end-1) );

        elseif n_perp == 1
            if length(t_perpinds) > 1
                display('Only one perp value, but multiple indices.  This
                    really shouldn''t happen!')
            end
            f = sum(v_N(parainds(t_perpinds{1})));

        else
            f = 0;

        end

        m_rdf(i) = f;
    end

end % parper_rdf()
```

B.3.6 Growth Rate Utility Script

Aaand the remainder is done in another sectioned script. The memory usage of this gets untenable for small timesteps: potentially hundreds of GB. Recoding it to not keep all data (only that which affects the current detector timeslice or whatever) would help. Making it cluster-deployable would be better, but that would take a good bit of work.

```
% Full distribution to growth rate code stack

%% Define the distribution

% Fiddle with topside distribution
fiddle = false;
if fiddle
    % Find unique energies for fiddling
    [ t_en, t_en_ind ] = uniquetol(r_mirror_EVPA(smap_EVPA.top.En,:), 0.001,
        'outputallindices', true);

```

```
    % Remove half of the energies
    t_en_find = vertcat( t_en_ind{1:2:end} );
    GR_dist = r_mirror_EVPA(:,t_en_find);
else
    GR_dist = r_mirror_EVPA;
end

GR_smap = smap_EVPA;

shortest_travel_time = min(GR_dist(GR_smap.time,:));
longest_travel_time = max(GR_dist(GR_smap.time,:));

density_const = 0.000314207783; % m_e*epsilon_0/e^2

% ionospheric background parameters
f_pe = 400000;
omega_pe = f_pe * 2*pi;
n_e = omega_pe^2*density_const;
iono_temperature = 2000; %in Kelvin

s_dyn_iono = struct('n', n_e, 'temp', iono_temperature, 'shift', 0, ...
    'PAcenter', [], 'PAwidth', []);

% secondary background parameters
maxw_temperature = 200000; % in Kelvin
t_temp_eV = maxw_temperature/11604.505;
% linearly interpolate n(T) from Table 1b in Lotko & Maggs 1981
if t_temp_eV < 137.1
    t_n_lotkomaggs = (0.44 - 0.83)/(137.1 - 62.4)*(t_temp_eV-62.4) + 0.83;
else
    t_n_lotkomaggs = (0.42 - 0.44)/(220.1 - 137.1)*(t_temp_eV-137.1) + 0.44;
end
maxw_particles = t_n_lotkomaggs * 1000000; % cm^-3 -> m^-3
maxw_shift = 0;
maxw_PAcenter = [];
maxw_PAwidth = [];

s_dyn_bg = struct('n', maxw_particles, 'temp', maxw_temperature, 'shift',
   maxw_shift, ...
    'PAcenter', maxw_PAcenter, 'PAwidth', maxw_PAwidth);

% beam parameter sets
% run 4 longest travel time is <14s
s_dyn_beams = {
    struct('n', 0, 'dwell_time', 5), ...
    struct('n', maxw_particles/50, 'dwell_time', 0.100, 'temp',
       maxw_temperature/5, 'shift', 400, 'PAcenter', [], 'PAwidth', []), ...
```

```
...%    struct('n', maxw_particles/5, 'temp', maxw_temp/16, 'shift', 300,
    'PAcenter', [], 'PAwidth', []), ...
    struct('n', 0, 'dwell_time', 5)
    };

% Builder function eats launch time, sample time,
% and the three dist structures.
s_dyn_dist = build_dyn_struct(0.001, 0.010, s_dyn_iono, s_dyn_bg,
    s_dyn_beams);

%% Sanity-check plots

java_numFmt = java.text.DecimalFormat;

% sort by energy for plotting, but we feed
% dynamic_distribution velocities and PAs
[ s_En, si_En ] = sort(GR_dist(GR_smap.bot.En,:));

h = figure(7777);
clf(h)

set(h,'position',[ 10 500 300*n_beams 400]);
suptitle('Electron Distribution Functions')

n_beams = length(s_dyn_beams);
for i=1:n_beams

    [ t_dist, s_dist ] = dynamic_distribution(s_dyn_dist.times(i), GR_dist,
        GR_smap, s_dyn_dist);

    t_width = 0.90/n_beams;
    subplot('position',[ 0.05+t_width*(i-1) 0.17 t_width 0.70 ])

    plot(s_En,t_dist(si_En),'k', s_En,s_dist{1}(si_En),'g.', ...
        s_En,s_dist{2}(si_En),'b*', s_En,s_dist{3}(si_En),'rx');
    set(gca,'fontsize',12)

    xlabel('Energy [eV]')
    foo = get(gca,'xticklabel'); foo{end}=''; set(gca,'xticklabel',foo);
    if i==1
        ylabel('$f(|v|)$','interpreter','latex');
        t_xlim = xlim; t_ylim = ylim;
    else
        set(gca,'yticklabel',[])
        xlim(t_xlim); ylim(t_ylim);
    end

```

```
    if s_dyn_dist.beams{i}.n == 0
        legend('Combined', [ char(java_numFmt.format(s_dyn_dist.iono.temp)) ' K ionospheric BG' ], ...
            [ char(java_numFmt.format(s_dyn_dist.bg.temp)) ' K secondary BG' ])
    else
        legend('Combined', [ char(java_numFmt.format(s_dyn_dist.iono.temp)) ' K ionospheric BG' ], ...
            [ char(java_numFmt.format(s_dyn_dist.bg.temp)) ' K secondary BG' ], ...
            [ char(java_numFmt.format(s_dyn_dist.beams{i}.temp)) ' K, ' ...
                char(java_numFmt.format(s_dyn_dist.beams{i}.shift)) ' eV-shifted beam'])
    end
end

%print('-dpng',[file_outdir '\topdist.png'])

%% azi_sum all top timesteps
launch_time = s_dyn_dist.launch_dt;
n_beams = length(s_dyn_dist.beams);

tic
dt_c_EVPN = cell(n_launchsteps,2);

v_launchsteps = 0:launch_time:s_dyn_dist.times(end)-launch_time;
n_launchsteps = length(v_launchsteps);

for i=1:n_launchsteps
    t_time = v_launchsteps(i);

    [ m_EVPN, smap_EVPN ] = time_azi_sum_chain (t_time, GR_dist, GR_smap, s_dyn_dist);

    t_strike = m_EVPN(smap_EVPN.time, :) + t_time;

    dt_c_EVPN{i,1} = m_EVPN;
    dt_c_EVPN{i,2} = t_strike;

end

dt_m_EVPN = [dt_c_EVPN{:,1}];
dt_v_EVPNt = [dt_c_EVPN{:,2}];
toc

%% Set up for perp_sum
% Now we go through and filter, for bottom time t-deltat to t
```

```
sample_time = s_dyn_dist.sample_dt;
v_timesteps = shortest_travel_time:sample_time:(n_beams*t_dwell)-launch_time;
n_timesteps = length(v_timesteps);

% Find the field-aligned velocities, for use as the
% center points in the reduced distribution function.
display('Uniquetol...')
v_paravels = uniquetol(dt_m_EVPN(smap_EVPN.bot.v.para,
    dt_m_EVPN(smap_EVPN.bot.v.perp,:)==0));
display('...done.')

% Reverse to smallest magnitude first
v_paravels = sortmag(v_paravels);

% extend these to zero
n_para = length(v_paravels);
d_vpar = median(diff(v_paravels));
d_extrap = v_paravels(1):-d_vpar:0;

v_paravelx = [ flip(d_extrap(2:end)) v_paravels ];
n_paravelx = length(v_paravelx);

%% perp_sum all top timesteps

display('Running perp_sum()s...')
tic
dt_m_rdf = zeros(n_timesteps,n_paravelx);
dt_v_nrdf = zeros(n_timesteps,1);
for i=1:n_timesteps

    t_time = v_timesteps(i);
    % search for particles that have been 'detected' in this timeslice
    v_timeinds = find(dt_v_EVPNt <= t_time & dt_v_EVPNt >
        t_time-sample_time);
    m_particles = dt_m_EVPN(:,v_timeinds);
    dt_v_nrdf(i) = size(m_particles,2);
    m_particles(smap_EVPN.dist,:) = m_particles(smap_EVPN.dist,:) *
        launch_time/sample_time;

    if length(m_particles) < 1
        continue
    end

%    display(['Time ' num2str(t_time) ' found ' num2str(length(m_particles))
    ' particles.'])

```

```
    % reduce to parallel
    dt_m_rdf(i,:) = perp_sum(m_particles, smap_EVPN, v_paravelx);

    % growth rate
%    v_gRate = para_gRate(m_rdf);

end
toc

%%

for i=490:800 %1:n_timesteps

    t_time = v_timesteps(i);
    % search for particles that have been 'detected' in this timeslice
    [ t_time t_time-sample_time ]
    v_timeinds = find(dt_v_EVPNt <= t_time & dt_v_EVPNt >
        t_time-sample_time);
    length(v_timeinds)
end

%% Full gamma vs k & time plot

t_temp = s_dyn_dist.beams{2}.temp;
t_shift = s_dyn_dist.beams{2}.shift;
t_bg = 2000; % background ionospheric cold electron temperature [K]
v_bg = sqrt(3*t_bg*15156333.1);

f_pe = 400000; % 500 kHz plasma freq.
omega_pe = f_pe * 2*pi;
t_test_temp = t_shift + t_temp/11604; % approximate beam speed
t_test_vel = eV2mps(t_test_temp);
[ ~, i_test ] = min(abs(-v_paravelx-t_test_vel))
v_omega_test = (1.00001:0.00001:1.01)*omega_pe;
v_k_test = sqrt(2/3*(v_omega_test-omega_pe)*omega_pe/v_bg^2);
%v_k_test = logspace(-6,20,1000);
v_k_test = 0.1:0.001:0.5;
v_omega_test = v_k_test.^2*3/2*v_bg^2/omega_pe + omega_pe;
n_test = length(v_k_test)

m_gamma = zeros(n_timesteps,n_test); % timestep,kind,val/omegaind
m_vtest = zeros(n_timesteps,n_test,2);
m_kmag2 = zeros(n_timesteps,n_test,1);
m_df1 = zeros(n_timesteps,n_test,1);
m_omega_test = zeros(n_timesteps,n_test,1);
m_n_e = zeros(n_timesteps,n_test,1);

```

```
parfor i=1:n_timesteps
    for j=1:n_test
        t_kpara = v_k_test(j);
%        t_omega_test = v_omega_test(j);

        [ m_gamma(i,j), m_vtest(i,j,:), m_kmag2(i,j), m_df1(i,j),
           m_omega_test(i,j), m_n_e(i,j) ] = growth_rate(dt_m_rdf(i,:),
           -v_paravelx, [ t_kpara 0 ], omega_pe, v_bg, t_test_vel);

    end
end

%% Launch timestep n summation, for 'this is where the beam was' plot.

dt_v_fbg = zeros(n_launchsteps,1);
dt_v_fbeam = zeros(n_launchsteps,1);
parfor i=1:n_launchsteps
    t_time = v_launchsteps(i);

    [ ~, s_dist ] = dynamic_distribution(t_time, GR_dist, GR_smap,
       s_dyn_dist);

    dt_v_fbg(i) = sum(s_dist{1}) + sum(s_dist{2});
    dt_v_fbeam(i) = sum(s_dist{3});
end


%% r/b gamma vs k,time plot

t_azi = 0;
t_el = 0;

v_upsteps = find(v_timesteps > 7.2 & v_timesteps < 8);
v_dnsteps = find(v_timesteps >= 6 & v_timesteps < 12);

%v_upsteps = find(v_timesteps);
%v_dnsteps = find(v_timesteps);

h = figure(7805);
clf
set(h, 'position', [100 50 1200 900])

p_plbase = 0.08;
p_plleft = 0.08;
p_plwidt = 0.40;
p_lpheig = 0.10;
p_vspace = 0.03;
```

```
p_spheig = 0.32;
p_hspace = 0.04;

hT = suptitle([ 'Growth Rates, $\Delta t_S =' num2str(launch_time) '$ s,
    $\Delta t_D =' num2str(sample_time) '$ s' ]);
set(hT,'interpreter','latex');

% -- n vs t --

nax = subplot('Position',[ ...
    p_plleft ...
    p_plbase+2*p_spheig+p_vspace ...
    2*p_plwidt+p_hspace ...
    p_lpheig ...
]); ax = [ ax nax ];
plot(v_launchsteps,dt_v_fbeam./dt_v_fbg)
xlabel('Time [s]'); set(gca, 'fontsize', 12); grid on; ylim([-0.25 0.75]);
    set(gca,'ytick',[0 0.2 0.4 0.6])
set(gca,'XAxisLocation','top');
    ylabel('$n_{beam}/n_{bg}$','interpreter','latex')
set(gca,'xtick',1:15)

% -- gamma vs t --

ax = [];
nax = subplot('Position',[ ...
    p_plleft ...
    p_plbase+p_spheig ...
    p_plwidt ...
    p_spheig ...
]); ax = [ ax nax ];
surf(v_timesteps(v_upsteps), v_k_test, m_gamma(v_upsteps,:).', 'edgecolor',
    'none'); colormap(rwbmap); box on;  set(gca, 'layer', 'top')
%caxis([-t_crange t_crange])
view(0,0); %set(gca,'zscale', 'log');  % ylim([min(v_k_test) 0.5])
%zlim([-10e3 10e3])
set(gca, 'fontsize', 12, 'xticklabel', []); ylabel('k'); zlabel('\gamma');
t_tick = get(gca,'ztick'); t_tick = t_tick(2:end); set(gca, 'ztick',t_tick);

nax = subplot('Position',[ ...
    p_plleft+p_hspace+p_plwidt ...
    p_plbase+p_spheig ...
    p_plwidt ...
    p_spheig ...
]); ax = [ ax nax ];
surf(v_timesteps(v_dnsteps), v_k_test, m_gamma(v_dnsteps,:).', 'edgecolor',
    'none'); colormap(rwbmap); box on; set(gca, 'layer', 'top')
```

```
t_crange = max([caxis(ax(1)) caxis(ax(2))]);
caxis(ax(1),[-t_crange t_crange]); caxis(ax(2),[-t_crange t_crange])
view(0,0); set(gca, 'ydir', 'reverse'); %set(gca, 'zscale', 'log');% ylim([min(v_k_test) 0.5])
zlim([-10e3 10e3])
set(gca, 'fontsize', 12, 'xticklabel', []); %ylabel('k'); zlabel('\gamma');
t_tick = get(gca,'ztick'); t_tick = t_tick(2:end); set(gca, 'ztick',t_tick);

% -- k vs t --

nax = subplot('Position',[ ...
    p_plleft ...
    p_plbase ...
    p_plwidt ...
    p_spheig ...
]); ax = [ ax nax ];
surf(v_timesteps(v_upsteps), v_k_test, m_gamma(v_upsteps,:).', 'edgecolor', 'none'); colormap(rwbmap); box on; set(gca, 'layer', 'top')
%caxis([-t_crange t_crange])
view(0,90); ylim([min(v_k_test) 0.5])
%ylim([0 3e-3])
set(gca, 'fontsize', 12,'xtick',get(ax(1),'xtick'))
ylabel('k');
t_tick = get(gca,'yticklabel'); t_tick{end}=''; set(gca, 'yticklabel',t_tick);
xlabel('Time [s]');

nax = subplot('Position',[ ...
    p_plleft+p_hspace+p_plwidt ...
    p_plbase ...
    p_plwidt ...
    p_spheig ...
]); ax = [ ax nax ];
surf(v_timesteps(v_dnsteps), v_k_test, m_gamma(v_dnsteps,:).', 'edgecolor', 'none'); colormap(rwbmap); box on; set(gca, 'layer', 'top')
t_crange = max([caxis(ax(3)) caxis(ax(4))]);
caxis(ax(3),[-t_crange t_crange])
caxis(ax(4),[-t_crange t_crange])
view(0,-90); set(gca, 'ydir', 'reverse'); ylim([min(v_k_test) 0.5])
%ylim([0 3e-3])
set(gca, 'fontsize', 12);  %ylabel('k')
t_tick = get(gca,'yticklabel'); t_tick{end}=''; set(gca, 'yticklabel',t_tick);
xlabel('Time [s]');

foo = annotation('line',[0.08 0.473],[0.72 0.749]);
set(foo,'color','red')
```

```
uistack(foo,'bottom')
foo = annotation('line',[0.48 0.528],[0.72 0.749]);
set(foo,'color','red')
uistack(foo,'bottom')

foo = annotation('line',[0.521 0.752],[0.72 0.749]);
set(foo,'color','red');
uistack(foo,'bottom')
foo = annotation('line',[0.92 0.808],[0.72 0.749]);
set(foo,'color','red');
uistack(foo,'bottom')

foo = annotation('textbox',[0.45 0.325 0.1 0.1], ...
    'string', '...', 'horizontalalignment', 'center', 'linestyle', 'none','fontsize', 16, 'fontweight', 'bold');

% print looks awful, use manual export
%print('-opengl','-dpng', [file_outdir '\gr.png'])

%% r/b gamma vs k,time plot, single zoom

t_azi = 0;
t_el = 0;

v_steps = find(v_timesteps > 7.2 & v_timesteps < 8);

h = figure(7805);
clf
set(h, 'position', [100 50 1200 900])

p_plbase = 0.08;
p_plleft = 0.08;
p_plwidt = 0.84;
p_lpheig = 0.10;
p_vspace = 0.03;
p_spheig = 0.32;

hT = suptitle([ 'Growth Rates, $\Delta t_S =' num2str(launch_time) '$ s, $\Delta t_D =' num2str(sample_time) '$ s, 100 ms Beam' ]);
set(hT,'interpreter','latex');

% -- n vs t --

nax = subplot('Position',[ ...
    p_plleft ...
    p_plbase+2*p_spheig+p_vspace ...
    p_plwidt ...
```

```
    p_lpheig ...
]); ax = [ ax nax ];
plot(v_launchsteps,dt_v_fbeam./dt_v_fbg)
xlabel('Time [s]'); set(gca, 'fontsize', 12); grid on; ylim([-0.25 0.75]); set(gca,'ytick',[0 0.2 0.4 0.6])
set(gca,'XAxisLocation','top'); ylabel('$n_{beam}/n_{bg}$','interpreter','latex')
set(gca,'xtick',[ 1:7 8:10]); xlim(v_launchsteps([1 end]));

% -- gamma vs t --

ax = [];
nax = subplot('Position',[ ...
    p_plleft ...
    p_plbase+p_spheig ...
    p_plwidt ...
    p_spheig ...
]); ax = [ ax nax ];
surf(v_timesteps(v_upsteps), v_k_test, m_gamma(v_upsteps,:).', 'edgecolor', 'none'); colormap(rwbmap); box on;  set(gca, 'layer', 'top')
t_crange = max([abs(caxis(ax(1)))]);
caxis([-t_crange t_crange])
view(0,0); %set(gca,'zscale', 'log');  % ylim([min(v_k_test) 0.5])
%zlim([-10e3 10e3])
set(gca, 'fontsize', 12, 'xticklabel', []); ylabel('k'); zlabel('\gamma');
t_tick = get(gca,'ztick'); t_tick = t_tick(2:end); set(gca, 'ztick',t_tick);

% -- k vs t --

nax = subplot('Position',[ ...
    p_plleft ...
    p_plbase ...
    p_plwidt ...
    p_spheig ...
]); ax = [ ax nax ];
surf(v_timesteps(v_upsteps), v_k_test, m_gamma(v_upsteps,:).', 'edgecolor', 'none'); colormap(rwbmap); box on; set(gca, 'layer', 'top')
%t_crange = max([abs(caxis(ax(2)))]);
caxis([-t_crange t_crange])
view(0,90); ylim([min(v_k_test) 0.5])
%ylim([0 3e-3])
set(gca, 'fontsize', 12,'xtick',get(ax(1),'xtick'))
ylabel('k');
t_tick = get(gca,'yticklabel'); t_tick{end}=''; set(gca, 'yticklabel',t_tick);
xlabel('Time [s]');

```

```
foo = annotation('line',[0.08 0.6805],[0.72 0.749]);
set(foo,'color','red')
uistack(foo,'bottom')
foo = annotation('line',[0.92 0.745],[0.72 0.749]);
set(foo,'color','red')
uistack(foo,'bottom')

foo = annotation('textbox',[0.45 0.325 0.1 0.1], ...
    'string', '...', 'horizontalalignment', 'center', 'linestyle',
       'none','fontsize', 16, 'fontweight', 'bold');
```