X-Ray Diagnostics for the Levitated Dipole Experiment

by

Jennifer L. Ellsworth

Submitted to the Department of Nuclear Engineering
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 2004

Author .....
Department of Nuclear Engineering
thesis date here

Certified by .....
Jay Kesner
Senior Scientist
Thesis Supervisor

Accepted by .....
Neil E. Todreas?
Chairman, Department Committee on Graduate Students

X-Ray Diagnostics for the Levitated Dipole Experiment

by

Jennifer L. Ellsworth

Submitted to the Department of Nuclear Engineering
on thesis date here, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Plasmas in the Levitated Dipole Experiment will initially be created using electron cyclotron heating and it is therefore expected that most of the plasma energy will be stored in the fast electrons ($T_e > 100$ keV). As a consequence of these fast electrons, substantial x-ray flux is expected. In the initial run campaign we plan to utilize two x-ray diagnostics. An x-ray pulse height analyzer will measure the energy spectrum of bremsstrahlung emission at four radially spaced locations. An x-ray camera [12], on loan from PPPL, will view the intensity of bremsstrahlung emission from a portion of the toroidal cross section of the plasma at sixty fields per second. Since the rapid toroidal drift of the hot electrons will symmetrize the hot electron component we expect that an asymmetry in the bremsstrahlung signal will indicate a spatial asymmetry in the ion population. We therefore expect to use the camera to indicate plasma asymmetries, which might indicate the presence of convective cells. The design, construction and calibration of these diagnostics will be discussed.

The x-ray pulse height analyzer and x-ray camera are vital to analyzing the first plasmas generated in LDX. The data from the XPHA will be used to optimize the ECRH heating scheme. Once we are able to generate stable, high beta plasmas, the x-ray diagnostics can be used to study interesting transport phenomema.

Thesis Supervisor: Jay Kesner
Title: Senior Scientist

Acknowledgments

My acknowledgments...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The x-ray pulse height analyzer (PHA) is a core diagnostic for the Levitated Dipole Experiment (LDX). Initial experiments in the LDX will study hot electron plasmas with temperatures on the order of 100 keV. These hot electrons are expected to produce significant bremsstrahlung radiation in the x-ray spectrum. The PHA will provide time resolved spectral measurements of the x-ray energy along four chords in the plasma. When combined with the equilibrium pressure profiles reconstructed from magnetic field measurements, the hot electron density and temperature may be determined. The hot electrons carry most of the energy in the plasma, so density and temperature will tell us how effective we are at coupling energy into the plasma.

LDX has been designed to study high beta, compressively stabilized plasmas in a dipolar magnetic geometry. The poloidal field is supplied by an internal superconducting magnet carrying a maximum of 1.5 MA. During levitated operation, the dipole field lines form closed loops which provide toroidal confinement without toroidal fields [8]. This magnetic geometry is similar to that of planetary magnetospheres [**?**].

The LDX is an attractive confinement concept for fusion because it is inherently capable of steady state operation. In addition, an LDX reactor require advanced fuels, eliminating the need for expensive materials and breeding technologies. The development of an LDX reactor would require the development of high field, high temperature levitating superconducting magnets[10]. To this end, the LDX is the first fusion experiment to use a high temperature superconducting magnet.

The purpose of the LDX is to understand the equilibrium, stability and confinement properties for a plasma that is confined in the field of a levitated dipole [**?**]. This can be broken down into two main parts: the study of high beta plasmas stabilized by compressibility, and the study of magnetic shear free systems.

Conventional magnetic confinement devices such as tokamacs rely on the toroidal field for confinement and the compressibility term is nearly zero. These types of devices are stabilized by magnetic shear and good curvature. In contrast, LDX has purely poloidal field lines, no magnetic shear, and a non-zero compressibility. Interchange modes are predicted to be stabilized by compressibility as long as the pressure falls off more slowly than V^{γ}, where V is the flux tube volume, $V = \oint \frac{dl}{B}$, and $\gamma = 5/3$. For a point dipole these conditions are satisfied for $P \propto R^{-20/3}$ [5]. The purely poloidal field has several other advantages. There is no particle drift off of the flux surfaces so the confinement can be nearly classical. The absence of magnetic shear is expected to decouple particle and energy confinement. In addition the device can operate in steady state without current drive.

In two dimensional systems such as LDX, convective cells may form, and transport can take place via these macroscopic flows[9]. For stable pressure profiles with "good curvature", convective cells would transport energy down the pressure profile, whereas for "bad curvature", they would transport energy up the pressure profile. If the pressure profiles are marginally stable with respect to interchange modes and stable with respect to all other modes, theory predicts that no energy will be transported by convective cells. If this condition is satisfied, then LDX will be able to operate at marginal stability without reducing the energy confinement time. LDX is currently the only fusion type experiment designed to operate at marginal stability.

An x-ray pulse height analyzer (PHA) is a plasma diagnostic that measures the energy spectrum of bremsstrahlung emission from the plasma. Bremsstrahlung, or "breaking radiation", is emitted when a free electron interacts with the electric field of a charged particle [6]. For initial plasmas in LDX, electron–ion as well as electron–neutral interactions will contribute to the bremsstrahlung spectrum. X-rays are also emitted when hot electrons collide with surfaces, such as the vacuum vessel walls

or the F-Coil. It is necessary to shield detectors from hard target bremsstrahlung with careful collimation to obtain electron temperature measurements from the PHA measurements.

The simplest form of a pulse height analyzer is an x-ray detector that produces a charge proportional to the energy of the incident x-ray. The charge is then converted to a voltage by a charge sensitive preamplifier. Each voltage pulse is shaped, filtered, digitized, and counted. The result is an energy spectrum of x-rays incident on the detector.

The PHA for LDX views four chords along the midplane of the vacuum vessel, providing enough spatial resolution for qualitative profile measurements. The diagnostic relies on 5x5x5 mm CZT (cadmium zinc telluride) detectors with energy ranges from 10 keV to 670 keV. Additional measurements can be made at higher energies using 2x2" NaI (sodium iodide) scintillation detectors which have an energy range of 1 keV to 3 MeV. The CZT detectors were selected for their small size and superior energy resolution.

Time resolution can be achieved by measuring multiple spectra during the course of a shot. At maximum energy resolution of 8184 bins, sixty-four spectra can be stored. The time windows during which each spectra is measured can be varied individually. Additional spectra can be measured before and after each shot. Time resolution was not implemented for the first plasma runs, but will be available for later runs.

For initial plasmas, the output of a sodium iodide detector, positioned to look directly at the F-Coil, was amplified and digitized. These measurements may provide some insight into the conditions during which x-rays emitted from the plasma. For example, when the LDX is operated at a reduced field with a small amount of heating power, it may take several seconds to produce a significant number of hot electrons.

The temperature of the hot electron population may be extracted from the x-ray spectra. This, combined with equilibrium pressure profiles reconstructed from magnetic measurements, can be used to determine the hot electron density. A qualitative picture of how the hot electron population is distributed within the plasma and how

that distribution evolves over time can be obtained. Understanding the hot electron population in the plasma is essential to understanding the LDX plasmas.

The goals for the pulse height analyzer are to look at hot electron plasmas with high beta to get an idea of the pressure profile kinetically. We would like qualitative measurements of the kinetic pressure profile to provide evidence for whether a particular pressure profile is marginally stable or not. We may be able to set some benchmarks for how plasmas behave in different magnetic field configurations.

We would also like to investigate the x-ray spectrum to determine the maximum energy, and number of x-rays emitted. This will indicate how efficient the hot electron production is, whether there are preferential hot electron losses at some energies and the upper limit of the electron energies.

The purpose of this thesis is to design, calibrate, and test the x-ray diagnostics for the LDX and to use measurements from these diagnostics to analyze the ECRH heating of first plasmas.

Chapter two will highlight important details of the Levitated Dipole experiment, including the magnetic geometry, initial diagnostic set, and typical plasma parameters. The PHA detectors were calibrated using an Am-241 source which has a calibration line in the range of the expected x-ray emission from LDX plasmas. The details of the x-ray pulse height analyzer including the choice of detectors, collimation, the data acquisition system and calibration are described in chapter three. The last chapter will present the results of initial experiments in LDX.

Chapter 2

The Levitated Dipole Experiment

The Levitated Dipole Experiment (LDX) will study plasma confinement in dipolar magnetic fields. The idea of using a dipole field for a fusion reactor was developed by Akira Hasegawa after the Voyager II spacecraft detected plasma trapped in the magnetic field of Jupiter's magnetosphere [**?**]. In nature, dipole magnetic fields have been observed to confine plasmas around neutron stars and middle magnetospheres of magnetized planets such as Jupiter and Earth [8].

Currently, the collisionless terrella experiment (CTX) is investigating supported dipoles [**?**]. Mini–RT in Japan is studying cold, single species plasmas with a 6 cm diameter levitated dipole. The LDX, in contrast, has a 68 cm diameter floating coil, and will initially study hot electron (neutral) plasmas.

This chapter will present a brief overview of the experiment, including the magnetic geometry, heating and diagnostic set.

2.1 Magnetic geometry

The magnetic geometry of LDX is achieved using three superconducting magnets. The dipole magnetic field is provided by an internal floating coil (F-Coil) carrying 1.5 MA. During floating operation, the 550 kg F-Coil will be continuously levitated from above using the levitation coil (or L-Coil). In supported mode, the L-Coil may also be used to provide some plasma shaping. The F-Coil is inductively charged by a

4 MA charging coil (C-Coil). Fig. 2-1 shows a cutaway view of LDX with the three magnets and the field lines for levitated mode with the L-Coil at full field. Additional plasma shaping can be achieved by imposing a vertical magnetic field with two copper Helmholtz coils.

Initial plasmas in LDX will use a supported dipole configuration. The F-Coil will be energized to 40% of the full current, 0.6 MA. Equilibrium field lines without shaping are show in figure 2-2.

2.2 Plasmas

LDX plasmas will be hydrogenic. Initial experiments were performed using deuterium gas as fuel. Hydrogen, helium and argon are also available for experiments on mass or charge scalings. Argon plasmas can be used to enhance x-ray signals because bremsstrahlung emission is proportional to Z^2 and argon has a much higher Z than hydrogen or helium.

Initial plasmas will be created using multifrequency electron cyclotron resonance heating (ECRH) at 2.45 GHz and 6.5 GHz. Each source produces 3 kW of power. 10 GHz will be incorporated in future run campaigns. The particle confinement time is expected to be good, so electrons heated by ECRH should reach high temperatures, on the order of 100 keV. In order to reach these temperatures using the configuration for first plasmas, shots on the order of ten seconds may be required.

The resonance zones for initial plasma experiments without shaping are shown in Fig. 2-2. The outermost line of each zone, 140 Gauss and 370 Gauss for 2.45 GHz and 6.5 GHz, are the cold plasma resonances for each frequency. As the electrons become relativistic, their lab frame mass increases and the resonance shifts inward. The inner edge of the resonance zone pictured is the second harmonic for 200 keV electrons. The red line marks the edge of the vacuum vessel. The F-Coil is pictured in the center in white.

There are two 'knobs' on the initial experiments. Varying the power of the ECRH sources will adjust the plasma profile. The relative power of the two sources can also

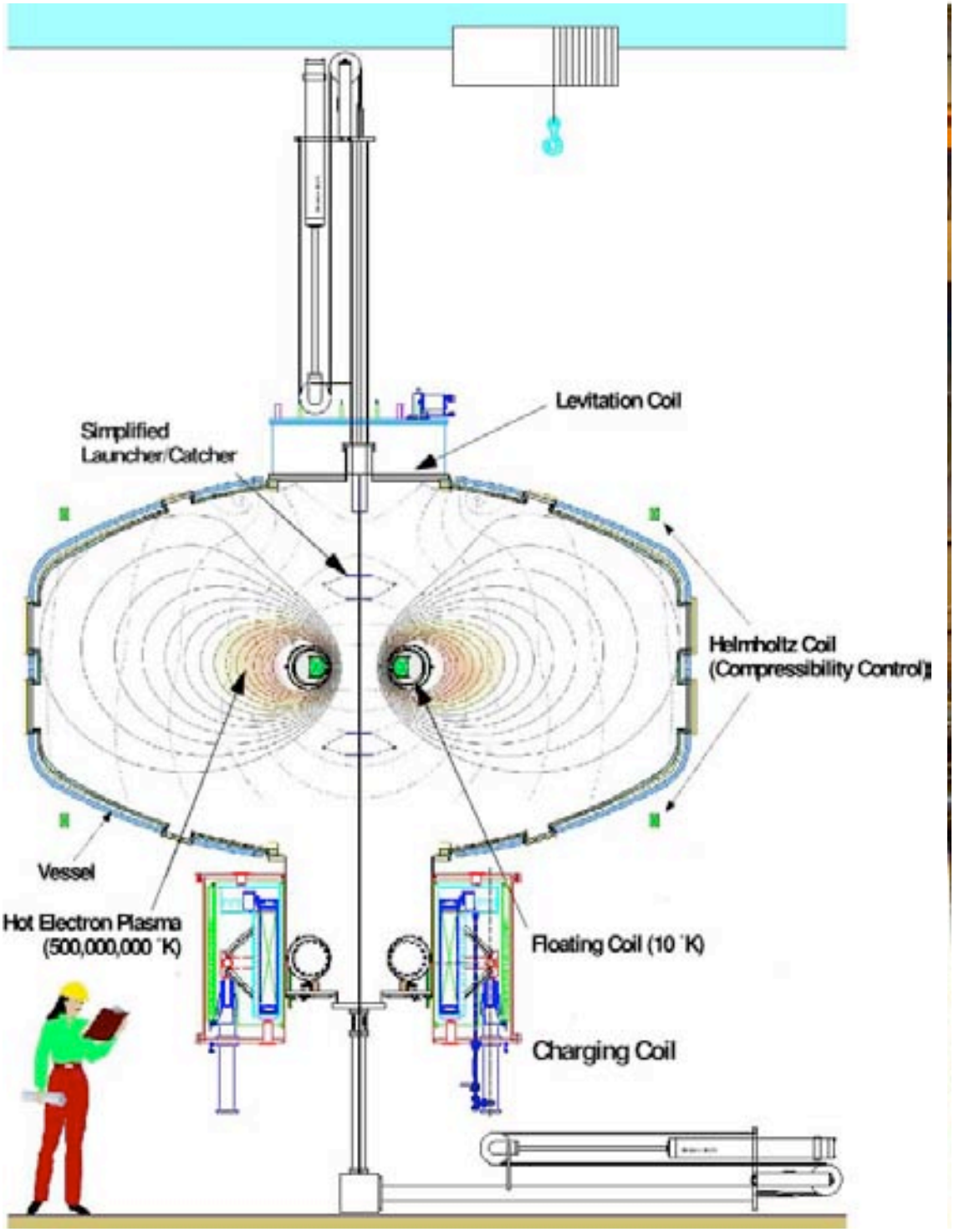

Figure 2-1: A cross section of the LDX experiment is shown with the basic coil configuration. Magnetic field lines (solid) and $|B|$ contours (dotted) are drawn for the plasma equilibrium resulting for levitated operation.

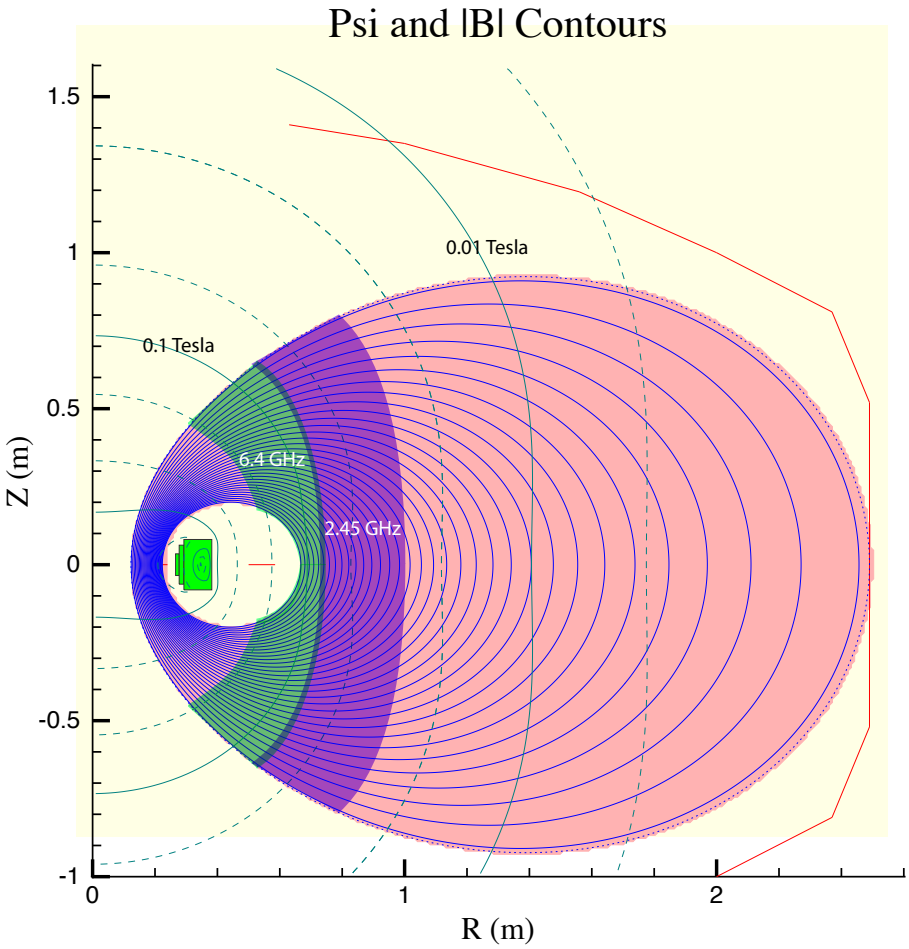

Figure 2-2: Typical magnetic geometry for first plasmas.

be varied..

The second control is the gas pressure. Initial experiments included a gas puff scan, during which deuterium gas was puffed into the vessel over times ranging from 2 to 100 ms.

An additional knob will be used in the future. The shape of the plasma can be manipulated by applying a vertical magnetic field via the Hemholtz coils or by energizing the L-Coil with a small amount of current. Plasmas can be shaped to a limited mode or a diverted mode.

The initial plasmas discussed in this thesis will use a supported dipole configuration with no shaping. The F-Coil current is 40% of the design current, approximately 0.6 MA so the maximum poloidal magnetic field is 0.022 Tesla. Data analysis is ongoing. Plasma density is predicted to be a maximum of $10^{11} cm^{-3}$. Edge densities are expected to be in the range of 10^{10} cm^{-3}. Hot electron electrons were observed in the range of 20 keV to 150 keV. The bulk ion and electron populations will have temperatures should be in the low eV range.

An estimate of plasma parameters for plasmas created using the F-Coil at full field with no shaping, limited mode, and diverted mode are given in Table 2.1 for four configurations. These parameters are for plasmas in levitated mode. Plasma densities are expected to be in the range of $10^{19} m^{-3}$.

Table 2.1: Plasma equilibria parameters. (A) diverted, no shaping, (B) diverted, shaped for maximum beta, (C) diverted, shaped for minimum beta, (D) limited plasma.

	A	B	C	D
S-Coil Currents; I_{s1}, I_{s2} (kA)	0,0	1,12	50,50	3,12
Plasma Volume (m^3)	14	27	1.7	24
SOL Pressure (Pa)	0.25	0.25	0.25	0.1
Max Pressure (Pa)	1.35	1530	45	472
Plasma Current(kA)	3.2	16.4	0.39	5.78
Stored Energy (J)	315	1450	27	516
R(P_{max}) (m)	0.76	0.76	0.77	0.79
B(P_{max}) (T)	0.088	0.088	0.088	0.088
$\beta(P_{max})$	0.08	0.55	0.015	0.15

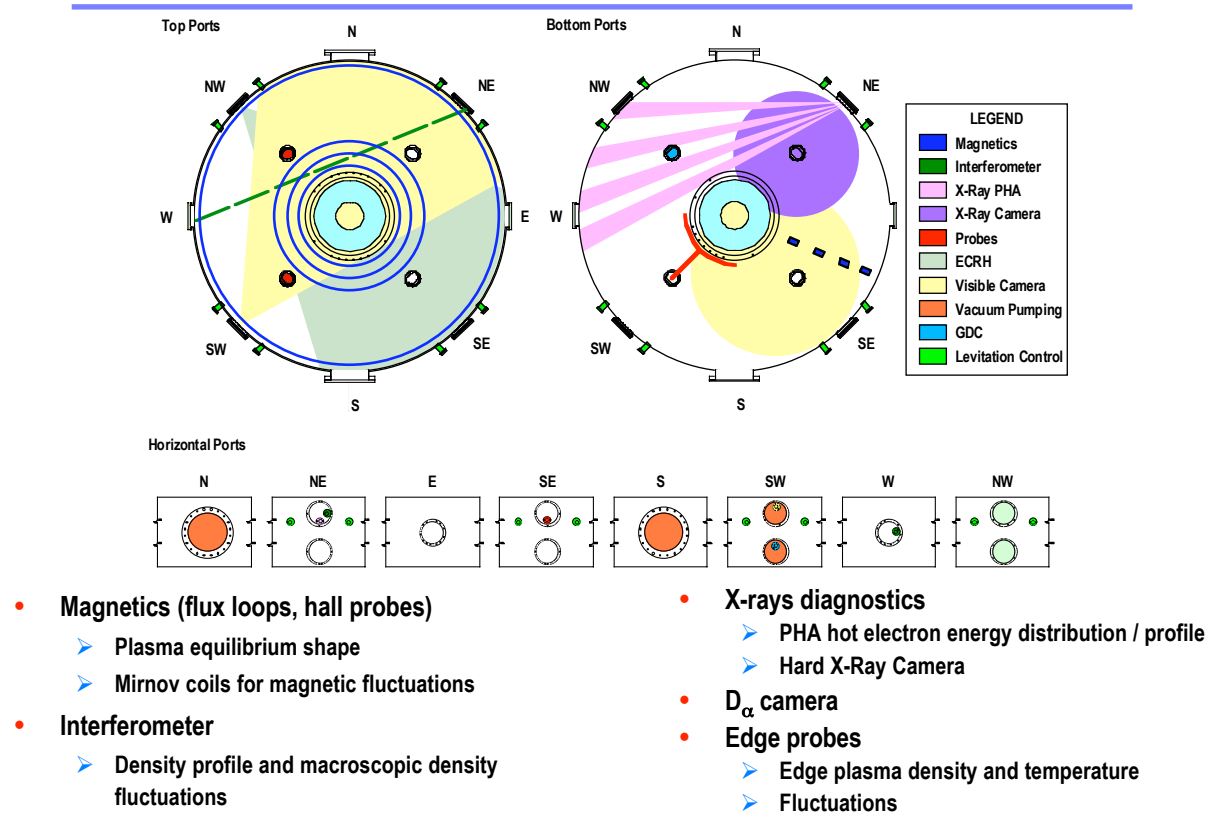

Figure 2-3: Placement of initial diagnostic set on LDX.

2.3 Diagnostic set

In addition to the x-ray measurements, the initial diagnostic set will include measurements of the pressure profile, plasma core density, edge density and hot electron temperature. Two black and white CCD cameras will give side and top views of the visible radiation and the launcher/catcher and positioning of the F-Coil. A digital video camera viewed from the side of the vessel. The locations of the initial diagnostics on the vacuum vessel are shown in Fig. 2-3.

- External magnetic measurements include 18 poloidal field coils spaced toroidally around the vacuum vessel and 6 flux loops. These will be used to reconstruct the MHD equilibrium pressure profiles. These measurements will have a temporal

resolution of 200 kHz.

- Internal magnetics will look at magnetic fluctuations potentially caused by interchange modes. One internal Mirnov coil will measure the magnetic fluctuations of first plasmas. This measurement also has a temporal resolution of 200 kHz. Seven additional Mirnov coils and two additional flux loops will be installed at a later date.

- A one-channel heterodyne interferometer operating at a frequency of 60 GHz will measure the core plasma density. Additional channels will be installed after the prototype is tested. This measurement was not digitized for initial experiments, but estimates based on observation of fringes on the oscilloscope are available.

- A four-channel x-ray pulse height analyzer measures the x-ray emission from the plasma during the shot. A single sodium iodide detector viewing through the center of the vessel provides a time resolved measurement of the x-ray intensity at 200 kHz.

- Three fixed position Langmuir probes measure edge density fluctuations.

- A monochromatic visible light camera is positioned to view through a side window. A second camera views the vessel from above. Both cameras output a standard video signal (30 frames per second). The cameras will be used for qualitative observation of the launcher/catcher operation and the plasma shapes. A color digital video camera views from the side.

Chapter 3

Experimental Apparatus - PHA description

For LDX plasmas, bremsstrahlung emission is expected from interactions between free electrons and free ions as well as interactions between free electrons and neutral particles. Bremsstrahlung is also emitted when hot electrons collide with objects in the vacuum vessel, namely the F-Coil, the F-Coil supports and the chamber walls. The energy spectrum of this x-ray emission is measured with a four channel pulse height analyzer. The design, installation and calibration of this diagnostic is described in the following chapter.

3.1 Bremsstrahlung radiation

I want to make fig. **??** using the eq. code of first plasma runs.

Figure 3-1: Predicted pressure profile generated from equilibrium reconstruction code and the corresponding predicted classical bremsstrahlung emission profile for various densities.

3.2 PHA design requirements

The PHA system for LDX was designed based on predicted hot electron temperatures on the order of 100 keV. Predicted count rates for an uncollimated system are 10^8 counts per second. Ideally, the temporal resolution should be on the order of the energy confinement time, but it must be long enough that a sufficient number of counts are collected for statistical analysis. Detectors and preamplifiers must be able to operate in 500 Gauss magnetic field.

3.3 PHA Layout

The simplest form of a pulse height analyzer is an x-ray detector that puts out a charge proportional to the energy of the incident x-ray. The charge is then converted to a voltage by a charge sensitive preamplifier. Each voltage pulse is then shaped, filtered, digitized and then counted. A block diagram of the pulse height analyzer designed for LDX is show in Fig.3-2. Each portion of the PHA will be discussed.

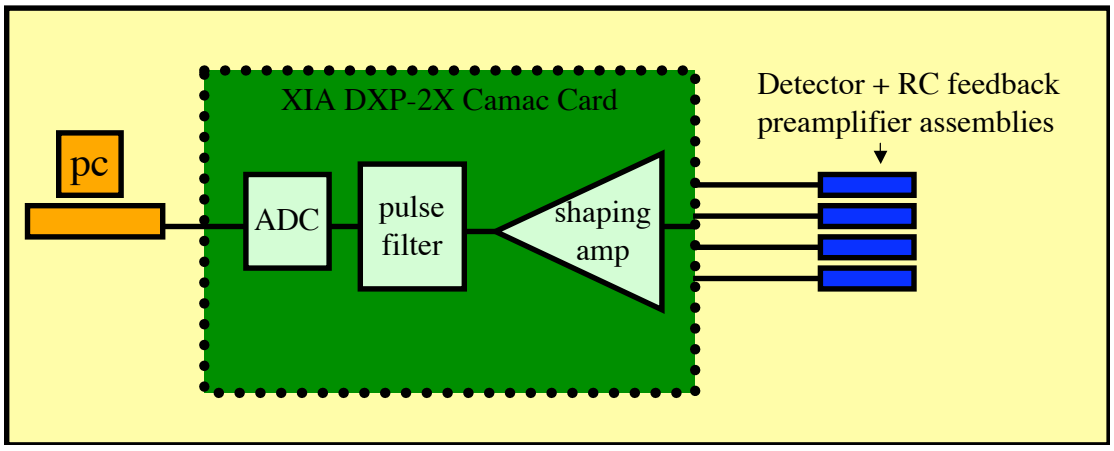

Figure 3-2: Black box diagram of the pulse height analyzer

3.3.1 Detectors/preamplifiers

Typical detector materials include silicon, sodium iodide (NaI), mercuric iodide, cadmium telluride (CdT), cadmium zinc telluride (CZT). The type of detector chosen

for each application depends on a number of considerations including the resolution, energy range, maximum count rate, and cost. Better resolution often can be obtained by cooling crystal detectors. Some detectors such as silicon detectors must be cooled to liquid nitrogen temperatures.

There are two types of charge sensitive preamplifiers, reset preamplifiers and RC feedback preamplifiers. In reset preamplifiers, the voltage is reset to zero between pulses. They are more difficult to build than feedback preamplifiers and are therefore more costly. During the reset there is a short 'dead time' during which no data can be collected. This type of preamplifier tends to be used with detectors for experiments that view soft x-rays and require high accuracy. An RC feedback preamplifiers is an RC circuit so that the voltage decays exponentially with time constant $\tau = RC$. Features are added to reduce noise and to correct for undershooting. RC feedback preamplifiers are inexpensive, but are noisier than reset preamplifiers.

Off the shelf detector and preamplifier units are available and adequate for our purposes. These units are fairly inexpensive and don't require time consuming development. Two types of detectors will be available for experiments. There are four CZT 5x5x5 mm crystal SPEAR detectors from eV Products. There is one NaI detector, model IA–1378 from Bicron with a 2x2" crystal. The NaI detector has a larger energy range than the CZT detectors, 5 keV – 3 MeV compared to 10 keV – 670 keV whereas the CZT detector has better resolution than the NaI detector, 4% full width half max at 122 keV versus 7% full width half max. The CZT detectors are also much smaller in size than the NaI detectors.

The digitizer has four channels so only four of the five detectors may be used simultaneously for pulse height analysis. Most of the emission is expected to be in the range of the CZT detectors, 10 keV – 670 keV [2]. Experiments from the collisionless terella experiment (CTX) at Columbia University show x-ray emission at energies as high as several MeV[1] so the NaI detector must be able to measure the overall x-ray emission from the LDX experiment so that we can be make sure we are not ignoring a high energy tail.

The Spear CZT detector can operate in magnetic fields up to 5 Tesla. The window

is 0.001" aluminum and 0.005" stainless steel, and the detector has a gold shield. The detector must be shielded from ultraviolet radiation. The NaI detector has μ metal shield and an aluminum window. Both detectors are shielded from visible light.

The smaller surface area of the CZT detector is not expected to be a problem. With 25 mm^2 area, $8x10^6$ cps are expected with no collimation. Although, the CZT is detector is much smaller than NaI detector, it is 50% more dense than NaI so has better count rate per unit volume.

The detector unit was chosen based on the preamplifier qualities. It was necessary to choose a preamplifier with a fast rise time because we expect a high count rate. The predicted count rate, gain and fall time are used to determine the voltage range of the output of the detector. The output must fall within the range of the digitizer, 0–10 V for the DXP4C2X. It is best to use as much of this range as possible without going out of range to increase the signal to noise ratio.

Equation 3.1 relates the rate of change in voltage in time to the preamplifier characteristics assuming a constant count rate.

$$\frac{dV}{dt} = G * \frac{N}{s} * E_{ave} - \frac{V}{RC} \tag{3.1}$$

The first term is increase in voltage due to an incident x-ray. The second term is the exponential decay of the voltage due to the RC time constant. This leads to a maximum voltage of ,

$$V_{max} = G * \frac{N}{s} * E_{ave} * RC. \tag{3.2}$$

The maximum voltage rise for the CZT detector is 4.0 V, due to a count rate of 500 kcps (the maximum count rate that can be processed by the digitizer), an average x-ray energy of 100 keV, detector gain of 0.11 mV/keV, and a fall time of 725 μs. The built in RC feedback preamplifier in the spear detector can handle count rates up to 500 kilo-counts per second. The same limit exists for the pulse height analysis electronics so this value was used to determine the maximum output voltage. 4.0 V is within the accepted range of the DXP4C2X. The DC offset of the Spear detector is less than 100 mV so this will not affect the result significantly. The NaI detector has

Table 3.1: Comparison of the characteristics of the Bicron 13xx NaI detector to the eV Products SPEAR CZT detector.

Parameter	**NaI**	**CZT**
Type	RC feedback	RC feedback
Energy Range	5 keV - 3 MeV	10 keV - 670 keV
Resolution	7% FWHM ?	4% FWHM 122 keV
Output gain	0.6 mV/keV	0.11 mV/keV
Rise Time	0.2 μs	0.035 μs
Decay time	50 μs	725 μs
Noise Equiv. Charge	160 e^- @ Cin = 6pF	160 e^- for Ce source
Ave Step Size	60 mV	11 mV
Assembly dimensions	2 " x 9"	12 mm x 89 mm

a gain of 0.6 mV/keV and a decay time of 50μs so the maximum voltage assuming 500 kcps is 1.5 V.

The detector and preamplifier properties are compared in Table 3.1. The gain of the preamplifier is given in millivolts per kiloelectron volts indicating the height of the voltage pulse is proportional to the energy of the incident x-ray. The rise time is the time it takes the voltage to spike when an x-ray hits. The fall time or decay time is the time it takes the voltage to drop to $\frac{1}{e}V_{peak}$. The noise equivalent charge is the charge in silicon of noise. The average step size, voltage increase as a result of an incident x-ray, has also been computed in millivolts.

3.3.2 Data acquisition and electronics

Traditionally, the shaping, filtering and counting of pulse is done with analog electronics. A shaping amplifier takes the preamplified pulse and shapes it into a gaussian peak, where the height of the gaussian is proportional to the energy of the incident x-ray and the width of the gaussian is related to the rise time of the preamplifier. Sometimes peaks pile up if incident x-rays hit the detector too quickly. Some pile up can be reduced by trying to fit a double gaussian to the pulse if a regular gaussian does not produce a good fit.

Analog systems are large and costly. We've chosen instead to use a multichannel

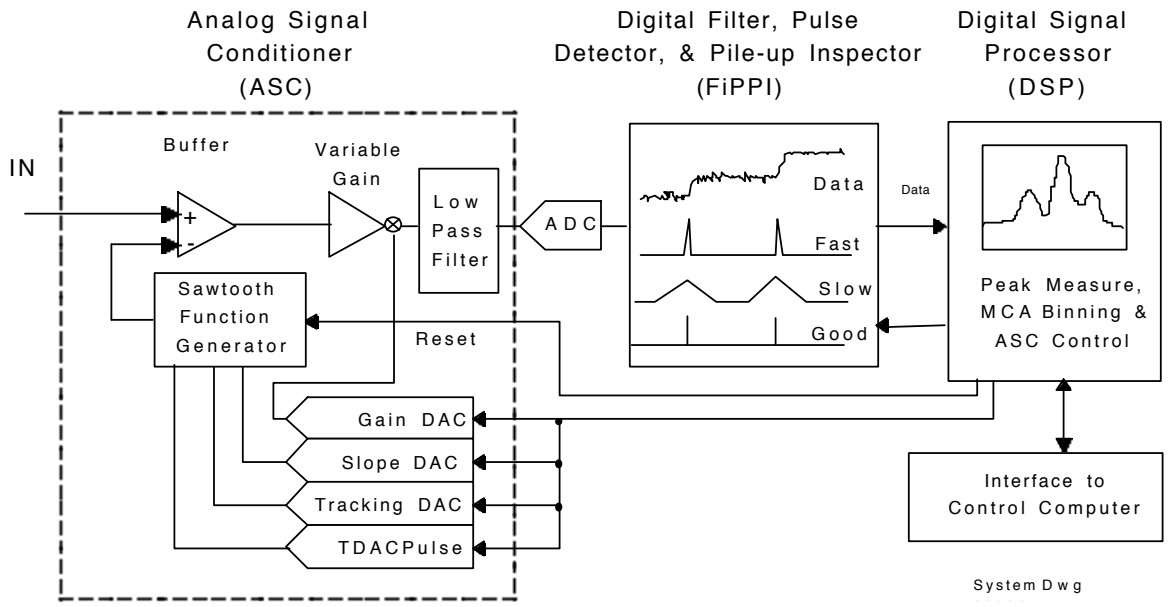

Figure 3-3: Block diagram of the function the XIA Digital X-Ray Processor card. Reproduced with permission from XIA.[13]

analyzer card, the Digital X-Ray Processor, DXP4C2X from X-Ray Instrumentation Associates (XIA), which runs on the CAMAC standard. The preamplifier signal is conditioned in an analog signal conditioner, then digitized. The digital signal is then shaped, filtered and counted. The energy spectrum is read out from the DXP4C2X to the computer.

Digital pulse height analysis has several advantages. The input pulse is already digitized when it reached the filters, so filtering and binning can be done in during the time of the pulse, thus reducing dead time and allowing for higher count rates. In addition, digital filters have a sharp termination which increases the throughput of the system, allowing for measurement of higher count rates.

3.3.3 DXP4C2X Multichannel Analyzer Camac Card

Rather than digitizing the preamplifier output directly, the DXP2X digitizes the difference between a known sawtooth function. This signal can then be digitized using a 10 bit digitizer. The signal then passes through a low pass filter to the analog

to digital converter. This unit is known as the analog signal conditioner (ASC). The sawtooth function can be adjusted while the system is running in the event that the input goes out range of the ASC.

The DXP2X uses two fixed length, trapezoidal filters to discriminate pulses, a fast filter and a slow filter. The voltage increase output from the preamplifier due to an x-ray incident on the detector is determined by taking the difference of the average voltage just after the pulse and the average voltage just after the pulse, as defined in Eq. filter

$$LV_{x,k} = -\sum_{i=k-2L-G+1}^{k-L-G} v_i + \sum_{i+k-L+1}^{k} v_i \tag{3.3}$$

Here, v_i is the voltage digitized at point i. L is the length time over which data is average before the pulse. Data is again averaged over a length L after the pulse. G is the gap, the rise time of the pulse. The filter lengths are set by the user.

Table **??** shows the filter characteristics used for initial plasma experiments. The filter parameters are input to the DXP2X in either clock units, 25 μseconds for a 40 MHz clock, or decimated clock units, $2^{DEC} * CLOCK_SPEED$. The decimation can be 0,2,4,or 6 and is determined by the DXP4C2X based on the peaking time. The current LDX configuration uses a decimation of 2. The slow filters are measured in decimated clock units, while the fast filters are measured in clock units. SLOWLEN and FASTLEN are the lengths of the slow and fast trapezoidal filters, respectively. Similarly, SLOWGAP and FASTGAP are the flattop times of the trapezoidal filters. The total time of the filter is $2 * LEN + GAP$. PEAKINT is the peak interval time and is typically set to SLOWLEN + SLOWGAP. PEAKSAM is the time at which the pulse height is sampled. This parameter can vary from SLOWLEN to PEAKINT-1. This parameter should be set such that the maximum count rate is achieved. A scan of the parameter using the LDX setup with CZT detectors found that value of PEAKINT - 4 yielded the maximum count rates, but there was very little difference in the count rates when values from PEAKINT-3 to PEAKINT-5 were used. THRESHOLD is the threshold for the fast filter trigger. Pulses smaller than the threshold are not measured.

Table 3.2: fippi

Filter Parameter	Value	Time
SLOWLEN	12	1.2 μs
SLOWGAP	6	600 ns
FASTLEN	5	125 ns
FASTGAP	1	25 ns
PEAKINT	18	1.8 μs
PEAKSAM	13	1.3 μs
MINWIDTH	2	200 ns
MAXWIDTH	20	2 μs
THRESHOLD	64	15 keV

To ensure that the filters have been set properly and to measure the noise in the system, the DXP2X periodically measured the voltage when there are no events to process. This set of measurements is the baseline. If the filters are set properly, the baseline should be gaussian distributed with a standard deviation that reflects the noise in the system.

There is electronic noise in the system introduced by fluctuations in the power supplies, capacitance in the cabling and frequency sources in the surrounding area. The standard deviation of this electronic noise, σ_e, is typically less than $0.5mV$ in the LDX system. The fano noise is a property of the leakage current in the detector. For CZT detectors, the noise equivalent charge in silicon is 160 electrons. The pair creation energy in silicon is $3.63eV$ which contributes approximately $0.063mV$ when adjusted for the gain of the detector. If the mean position of the baseline is not zero, the the height of the input pulses are scaled using that as the zero. This introduces some additional noise into the system as shown in Eq.3.4.

$$\sigma_t = \left(\sigma_f^2 + (1 + 1/64)\sigma_e^2\right) \text{[13]} \tag{3.4}$$

Additional statistics are acquired for each run. These include the realtime, which is the total time that passed during the run. The livetime is the time during which the DXP2X was processing events, rather than performing maintenance operations

such as measuring statistics and the baseline or readjusting the ASC parameters. The rate of counts into the filters and rate of counts that meet the pile up criteria and are counted. The total number of counts in the run and the number of times the fast filter was triggered are also stored.

The DXP-4C2X has two options for collecting time resolved data. In list mode, the DXP2X records the time of a pulse and the height of the pulse. This information is stored on the card until the run is stopped or the memory is full. This configuration is limited by the total number of photons that can be collected each shot. Each channel has 1 MByte of onboard memory so 256k events can be stored. Livetime statistics are recorded for the entire list.

The second configuration is to take a sequence of spectra in time, multiMCA mode. Livetime statistics are recorded for each spectra independently so a more accurate measurement is obtained with this method than with list mode for applications where the x-ray signals very significantly with time. The time windows do not have to be of equal length.

Again, the number and size of the spectra are limited by the memory. We would like to have enough counts in each spectra for good statistics and enough spectra to see the time evolution of the signal on the timescale of the ECRH heating. We would also like to have the maximum number of bins in each spectra to provide the best energy resolution. This is important because x-ray emission is expected in the range from several keV to several MeV and each bin is the same size. The maximum number of bins is 8192. We chose to use 64 spectra of 8184 bins each. The length of the spectra is determined by an external gate signal applied from a Jorway 221 card. The DXP2X can operate with count rates up to 500 kcps. At the maximum rate, each bin would contain 7812 counts.

To take advantage of this multi-MCA mode, a firmware upgrade was required for the DXP4C2X. XIA developed and tested this firmware. The LDX will be the first to put this firmware to use. This portion of the system was not operational at the time of first plasma experiments.

3.3.4 DXP4C2X control software and driver

The benefit of using a multichannel analyzer card is that there is no need for costly analog electronics. All of the pulse height analysis is done on the card or in software. The software is provided by XIA. Unfortunately, the software provided does not support using a jorway 411s camac controller.

Two software packages are available from XIA to control the DXP2X. Mesa2X is a LabView program which allows the user to record and view spectra, baselines and statistics. A digital oscilloscope mode is available that shows the voltage trace of the preamplifier output and the outputs of the fast and slow filters. This program also includes an auto-calibration feature that calibrates the detectors to a single line source. Handel is a set of C libraries that provides the user with more flexibility than Mesa2X, but is not as automated.

Mesa2X was used for testing and calibration, while the Handel libraries were used for experimental runs. Both programs required the development of a driver to be used with the LDX CAMAC hardware. The LDX is using a CAMAC 411s controller which interfaces between the computer and the CAMAC crate controller. MDSPlus already includes a driver for the Camac 411s controller.

An interface was written between the two programs from XIA and MdsPlus, the data acquisition software used for the experiment. The interface takes a command from Mesa2x and calls an MDSPlus command on a user specified server from the remcam library which then executes the command using the built in driver. The camac controller is physically connected to the server which does not need to be the same computer that is running the Mesa2x software. The interface supports both VMS and Linux operating systems as the MDSPlus server. The driver source code is included in appendix A.

Using the Handel and MdsPlus libraries, additional programs were to control the initialization, triggering, and storage of data from the DXP4C2X into the MdsPlus tree. Programs were written for both single spectrum and time resolved mode. The source code for the single spectrum mode of operation is available in appendix B.

3.3.5 Views and collimation

The chords that the PHA will view were chosen such that the entire midplane of the plasma would be sampled. The chords were chosen to be equidistant in angle, rather than clustered where the pressure peak is expected, because we would like to investigate how varying the heating affects the pressure peak. We would like to have the flexibility to diagnose plasmas that do not behave as expected. Fig. 3-4 illustrates the detector views in the LDX vacuum vessel.

A second Be window views the center of the vacuum vessel. A fifth detector can view through this window if alternate signal digitization is used. The NaI detector was placed on this window for first plasma experiments. This channel provide an measurement of the intensity of x-ray emission from the plasma. This number can then be used to classify plasmas.

The collimator has been designed for four CZT detector assemblies with 12 mm diameter. The collimator features an adjustable viewing angle and a replaceable pinhole. The entire assembly was designed to be flexible in the amount of collimation provided so that the PHA could easily adapt to a wide variety of plasma conditions. In the event that the bremsstrahlung emission is weaker than expected the collimation angle can be increased to allow for more signal. The design angle of four degrees was selected so that a full view of the plasma midplane could be obtained. The collimator is made of lead. The shortest thickness of lead an unwanted photon must pass through to reach a detector is 1 inch.

For first plasma operations, a pinhole was not used because the x-ray flux was expected to be low in comparison to the values of x-ray flux predicted for levitated operation. The aperture is a single slit, as pictured in Fig. 3-6. The étendue, G of the optical system is defined as

$$G = A\Omega_s$$

, where A is the area of the detector and Ω_s is the solid angle of the of the collimating optics. When several optics are coupled together, the étendue is determined from the component with the smallest value. When a pinhole is added to the PHA, its size

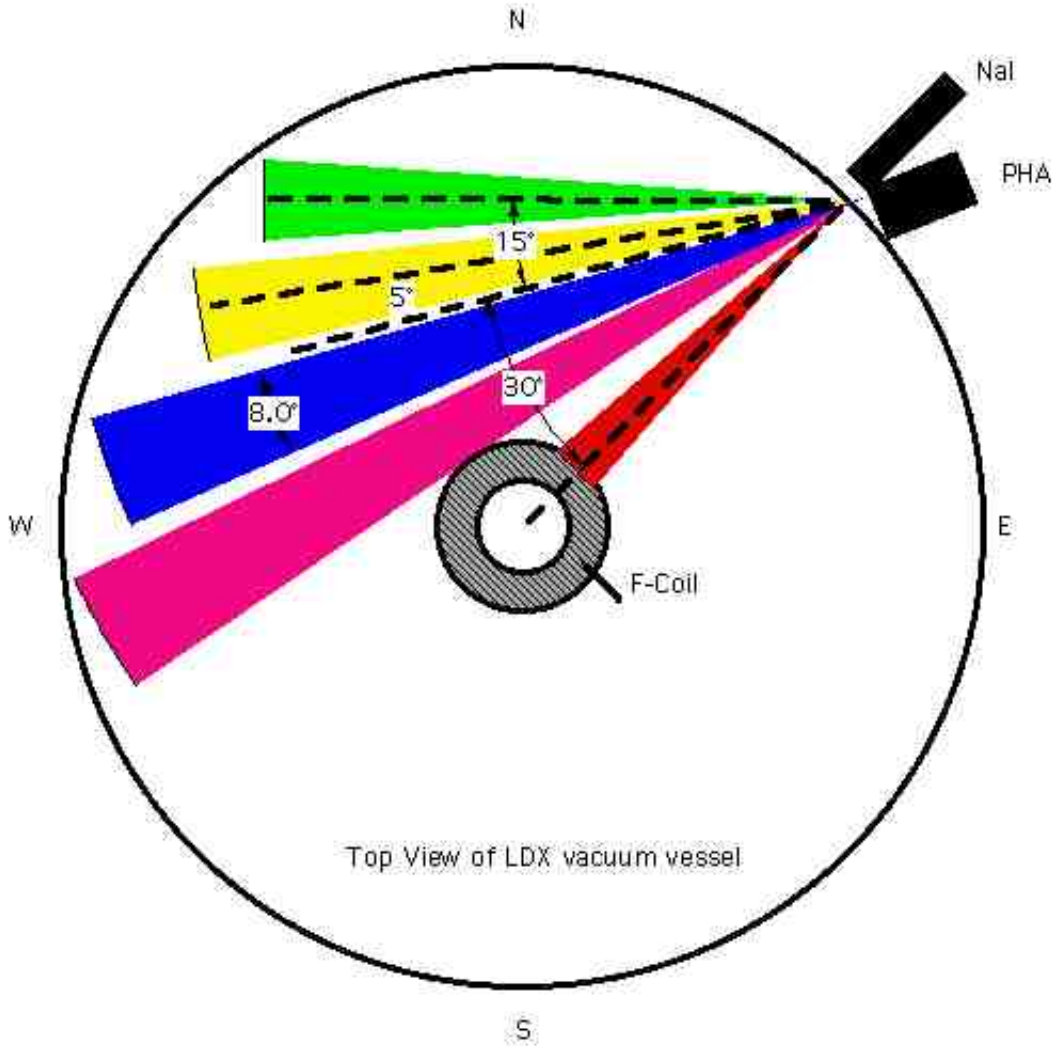

Figure 3-4: The chordal views of the Bremsstrahlung signal are shown overlayed on a cross-section of the LDX vacuum chamber. The floating coil is shown to scale in the center. An additional sodium iodide detector views directly across the vacuum vessel. The pressure peak is expected to lie close to the outer edge of the floating coil.

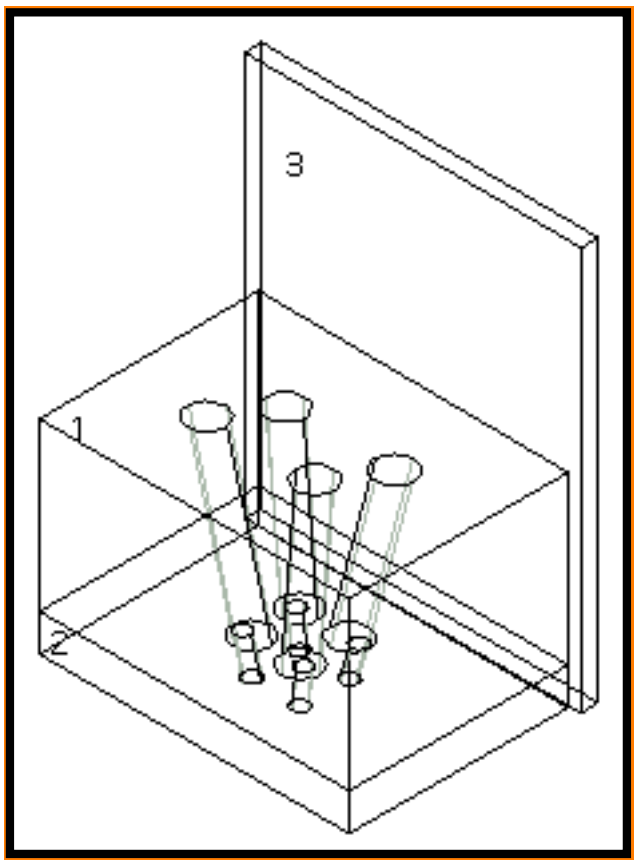

Figure 3-5: A schematic of the collimation device is shown on the left. A photograph is shown on the right.

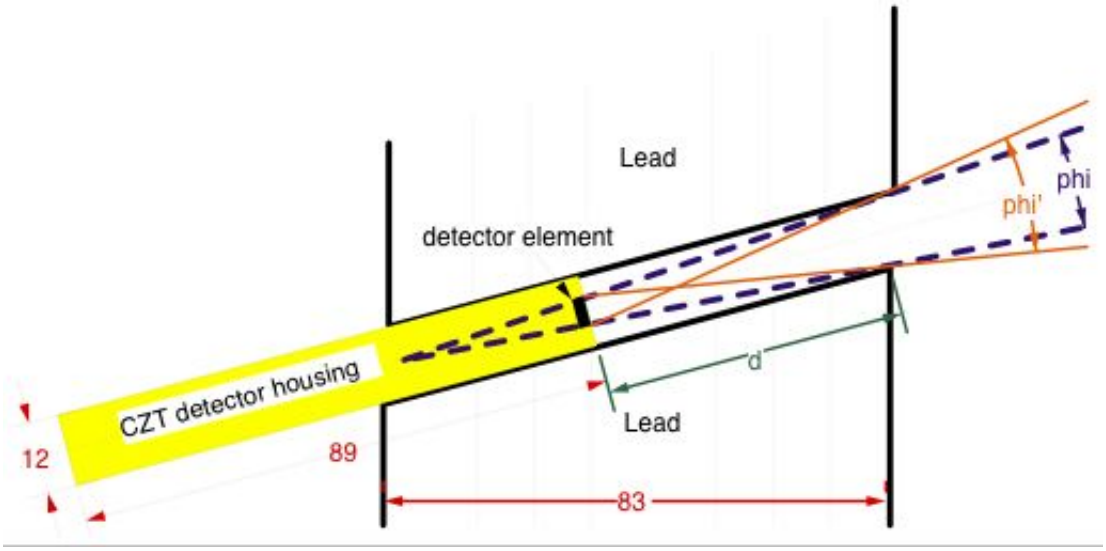

Figure 3-6: **The collimation setup for first plasmas. Each CZT detector was placed in a collimation hole. Adjusting the position of the detector changes the collimation angle.**

will determine the étendue of the system. For the experiments described here, the entrance hole to the collimator defines the étendue.

The detectors are oriented in the lead brick at $\pm 5°$ and $\pm 15°$. The collimator is symmetric about the center axis so we only need to calculate the optical properties for the two angles. The distance between the collimator opening and the detector is adjustable so it is necessary to solve for étendue in terms of this distance, d. The entrance hole is at a slight angle. This affects the viewing angle by about $0.1°$ for the distances between the detector and opening used in the experiments so we will neglect it for this calculation. The solid angle is

$$\Omega_s = \iint_S \sin\phi, d\theta, d\phi = 1 - \cos\phi_{max}$$

for cones. Let R be the radius of the collimator opening and r be the radius of the detector, then $\phi_{max} = \arctan(\frac{R-r}{d})$. So we can write

$$G = A(1 - \frac{d}{\sqrt{d^2 + (R-r)^2}})$$

. For small angles this simplifies to

$$G \simeq A\pi \frac{R^2 - r^2}{d^2}$$

The detector area is 5x5mm so $A = 25mm^2$. The collimator opening is 12.25 mm in diameter and a typical distance d used in the first experiments is 58 mm for a 15° angle. These values lead to a value of $0.73mm^2 steradians$ for étendue.

3.4 Calibrations

The detectors were calibrated using an Am-241 source. The calibration line for this source is 59.5412 keV[**?**]. The activity of the the source is such that count rates of 10^6 counts per second can be acheived. This is sufficient for testing pulse pileup characteristics. This line is in the low range of the LDX bremsstrahlung emission.

Table 3.3: CZT detector gains in mV/keV calibrated using and iterated gaussian fit to the 59.5412 keV line of Am-241. Zero is measured as the offset of the baseline measurement.

Detector Serial #	**DXP4C2X Channel**	**Detector Gain**
B1241	0	0.106486
B1242	1	0.102311
B1243	2	0.107559
B1244	3	0.103193

The detectors are calibrated using an iterated gaussian fit to the peak that is built in to the Mesa2X software. The time duration of each iteration is determined such that the height of the calibration peak is at least 1000 counts. Five iterations are used. The zero is set automatically in the software using the mean position of the baseline measurement.

The calibrated detector gain, baseline mean, baseline standard deviation for each channel are reported in Table 3.3. A typical Am-241 spectrum is as measured from one of the CZT detectors is shown in Fig. 3-7. The variation in the baseline mean is shown in Fig. **??**.

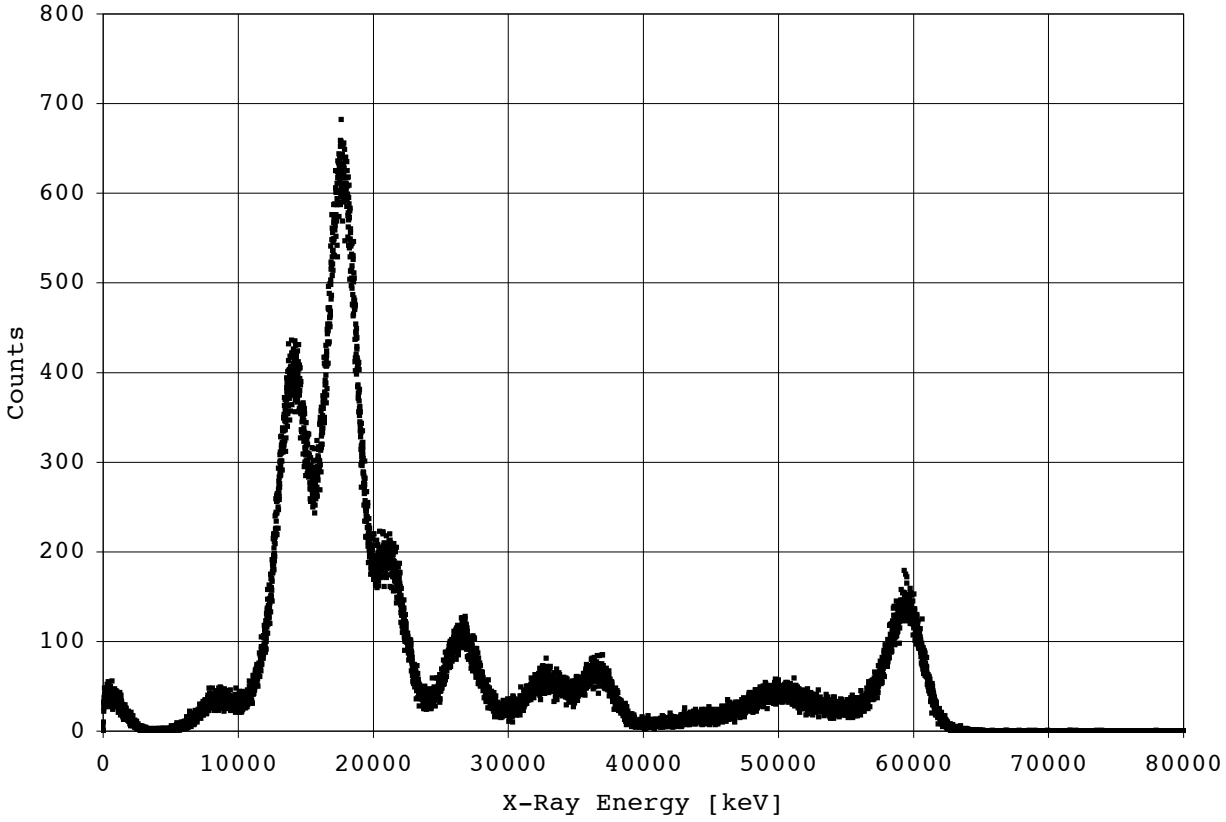

Figure 3-7: A typical Am-241 spectrum taken with a CZT spear detector using a threshold of 7 keV.

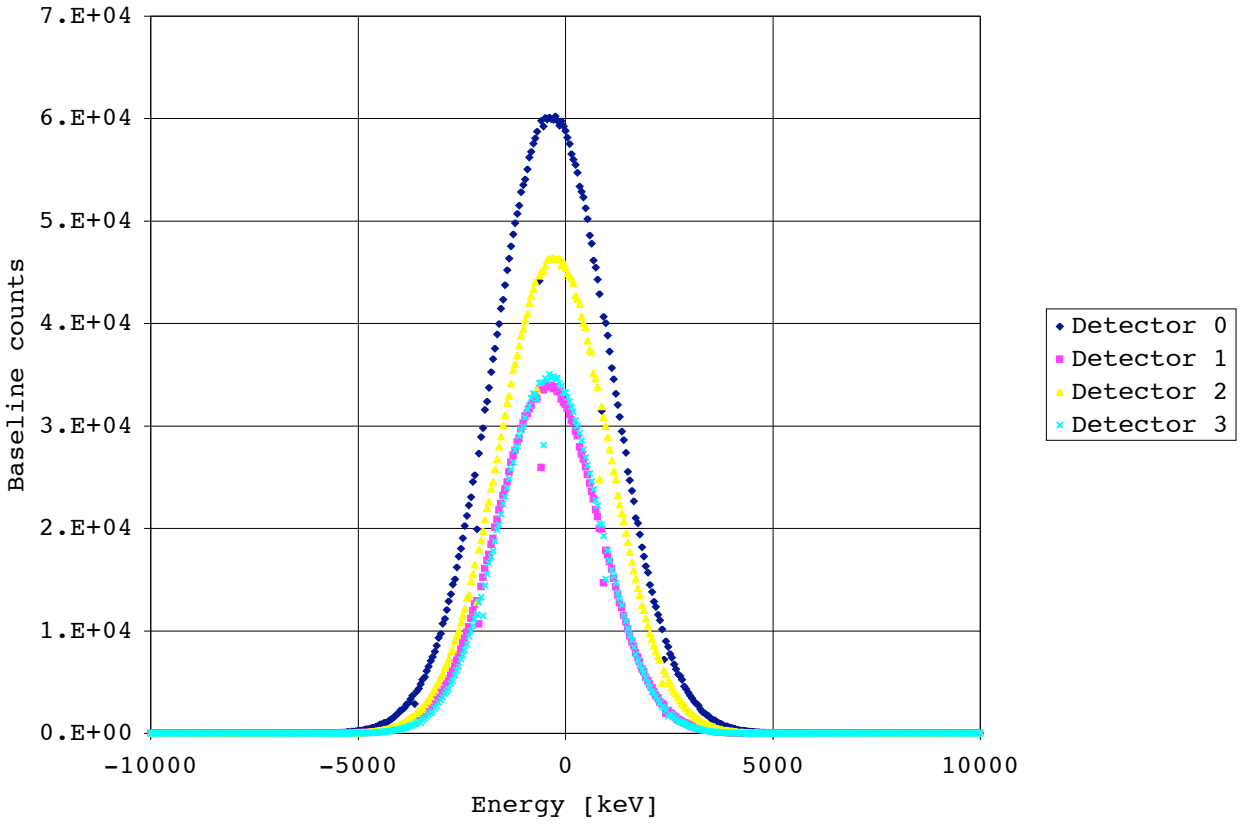

Figure 3-8: Typical baselines for CZT detectors are shown.

Chapter 4

Results from LDX

First plasma experiments on LDX were conducted Friday, August 13, 2004. These experiments utilized two frequencies of RF heating, 2.45 GHz and 6.5 GHz. Each source contributes 3 kW of power. RF power was not varied for these experiments. The heating times were varied from 4 s to 8 s. An RF only shot with the 2.45 GHz produced no plasma, suggesting that this source either wasn't firing or wasn't coupling energy into the plasma.

The base vacuum was in the low 10^{-7} Torr scale. A gas scan was conducted by puffing in deuterium gas for varied amounts of time.

Chapter 5

Conclusions

```
/*
 * mds_camacdll.c
 * 05/05/2004 Jennifer Ellsworth
 *
 * Revised version of machine_dependent_mds.c (created by Darren Garnier), a
 * driver for the XerXes library on which handel library is based.
 *
 * Uses function definitions from camacdll.c created by Ed Oltman.
 *
 * MdsPlus functions are taken from the MdsPlus remcam library.
 * http://www.mdsplus.org
 *
 * Interface between handel library and MDSPlus Camac 411s driver. Camac
 * routines are called using MdsValue("...") commands which call MDSPlus
 * library functions. Currently set up to use the MdsIpShr library by setting
 * macro EXCLUDE_MDS_LIB.  Otherwise, MdsLib is called. Currently, code only
 * works with MdsIpShr library.
 *
 * DXP4C2X module is addressed through a server and a string.  The server
 * servername: or local is the MDSPlus server to which the Camac
 * controller is physically connected and the string is the logical name
 * of the DXP4C2X module in the Camac crate.
 *
 * Driver has been tested using VMS and Linux systems as MdsPlus server.  VMS
 * servers use a different library for camac commands.  The driver will
 * automatically determine if the server is VMS or Linux and use the correct
 * command.
 *
 * MdsPlus specific configurations are stored in file INIFILE defined by
```

```
 * macro in mds_camac.h
 *
 * set Macro MESA2X to use with MESA2X program on windows or unset to use
 * with handel library.
 *
**************************************************************************/

/* Settings for this build */
/* #define DEBUG */
/* use this to override mode setting - for debugging */
/* #define FORCE_MODE_16BIT */

/* #define MESA2X 1 */ /* else assume handel is being used */

/*#ifdef MESA2X*/
/* This should get set in the Makefile for *nix systems */
#define EXCLUDE_MDSLIB
/*#endif*/

#include <stdio.h>

#ifdef MESA2X
#include <windows.h>
#else
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#endif

/* some important files from mdsplus... */
```

```
#include "ipdesc.h"
/* #ifndef EXCLUDE_MDSLIB */
/* #include "mdslib.h */
/* #include "mdsshr.h */

#include "mds_camac.h"
#include "camac.h"

#ifdef MESA2X
#define EXP __declspec(dllexport)
#else
#define EXP extern
#endif

EXP long mit_caminit(short* buf);
EXP long mit_camxfr(short* cam_addr, short cam_func , long len,
short mode, unsigned short* buf);

static char* read_word(FILE *fp);
static int read_ini_file(short *buf, int *nServers, int *nCrates, int *nCards);
static int CamSingle(int serverid, char *routine, char *name, int a, int f,
     void *data, int mem, unsigned short *iosb);
static int DoCamMulti(int serverid, char *routine, char *name, int a, int f,
      int count, void *data, int mem, unsigned short *iosb);
static void getdata(int serverid, void *data);
static int VMSCamSingle(int mdsSock, char *routine, char *name, short address,
short function, long count, void *data, int mem,
unsigned short *iosb);
static int VMSCamMulti(int mdsSock, char *routine, char *name, short address,
       short function, long count, void *data, int mem,
```

```
       unsigned short *iosb);
long MdsQuery(SOCKET socket, char *question);
int IsVMSServ(SOCKET socket);
int getNumWords(int nbytes, int mem);
int GetCamXandQ(SOCKET socket);
int GetCamX(SOCKET socket);
long GetCamBytCnt(SOCKET socket);
void LogData(struct descrip data);



#ifndef MESA2X
#ifndef MdsIpFree
#define MdsIpFree free
#endif
#define min(x,y) ((x) < (y)) ? (x) : (y)
#endif

typedef struct sCrate tCrate;
struct sCrate {
  int nCards;
  char *Card[MAX_CARDS];
};

char     gServerNames[MAX_ADAPTS][256];
long     gServerSocks[MAX_ADAPTS];
tCrate * gCrates[MAX_ADAPTS][MAX_CRATES];
int  gGoodCrates[MAX_ADAPTS][MAX_CRATES];
```

```
/* Define error logging functions.Errors are logged to file defined by  *
 * LOG_FILE environment variable in mds_camac.h                          */

#ifdef DEBUG
  static FILE *gLogFile = NULL;

  #define OpenErrorLog() do { gLogFile = fopen(LOG_FILE,"a");} while (0)

  #define CloseErrorLog() fclose(gLogFile)

  #define ErrorLog(format, ...) do {            \
    fprintf(gLogFile, format, __VA_ARGS__); \
    fflush(gLogFile);                          \
  } while (0)

  #define ASSERT(cond, string) if (cond) ErrorLog("Line %d: %s, %s\n",\
  __LINE__, #cond, string)

  #define LOG_CAMAC_STATUS(status) do {                                    \
    int _i, _max;struct stsText *_txt;                                     \
    _max = sizeof(camshr_stsText)/sizeof(camshr_stsText[0]);               \
    _txt = camshr_stsText;                                                 \
    for (_i=0; _i< _max; _i++) {                                           \
      if ((status & 0xfffffff8) == (_txt->stsL_num & 0xfffffff8)) {        \
        ErrorLog("Camac status: 0x%lx, %s\n",status,_txt->stsA_text);      \
        break;                                                             \
      }                                                                    \
      _txt++;                                                              \
    }                                                                      \
    if (_i==_max)                                                          \
```

```
      ErrorLog("Camac status: 0x%lx\t",status);                                \
  } while (0)

#else

#define OpenErrorLog()
#define CloseErrorLog()

#define ErrorLog(...)
#define ASSERT(cond, string)
#define LOG_CAMAC_STATUS(status)

#endif


/*****************************************************************************
 * Routine to initialize camac crate connectred to server and return number of
 * servers crates and cards.  Server is read from a fixed INI_FILE as defined
 * in mds_camac.h .
 *****************************************************************************/
static int read_ini_file(short *buf, int *nServers, int *nCrates, int *nCards)
{
  int nservers,ncrates,ncards, swapend;
  int i,j,k,index;
  char sectBuf[256];
  char    cardKey[256];
  int len;
  FILE *fp;
  char *pbuf;
  int bytes = 4;
```

```
  char error_string[132];
  int status;

  /* initialize array */
  for (i=0; i<MAX_ADAPTS; i++) {
    gServerNames[i][0] = '\0';
    if (gServerSocks[i]) {
      gServerSocks[i] = 0;
    }
    for (j=0; j<MAX_CRATES; j++) {
      if (gCrates[i][j] != NULL) {
free(gCrates[i][j]);
gCrates[i][j] = NULL;
      }
      gGoodCrates[i][j] = 0;
    }
  }

  /* open ini file */
  if ((fp = fopen(INI_FILE,"r"))==NULL)
    {
      sprintf(error_string, "Could not open file %s\n",INI_FILE);
      status = kErrBadMdsIniFile;
      ErrorLog("Could not open file %s\n",INI_FILE);
      return status;
    }

  sprintf(error_string,
  "Preparing to read the mdsplus specific ini file.  Filename = %s.\n"
  ,INI_FILE);
```

```
/* dxp_md_log_info("read_ini_file",error_string);*/
ErrorLog("%s", error_string);

/* read MDSPLUS_SECTION  from INIFILE file */
pbuf = read_env(fp);

if ( (strcmp(pbuf, MDSPLUS_SECTION)) != 0 )
  {
    ErrorLog("Error: Invalid INI file, %s. No entry for %s.\n",
     INI_FILE, MDSPLUS_SECTION);
    len = strcmp(pbuf, MDSPLUS_SECTION);
    ErrorLog("strcmp(%s, %s) = %hu\n", pbuf, MDSPLUS_SECTION, len);
    return kErrBadMdsIniFile;
  }

/* read number of mdsplus servers from inifile */
pbuf = readKey(fp, KEY_N_SERVERS);
if (pbuf != NULL)
  nservers = atoi(pbuf);
else
  ErrorLog("Error reading %s. readKey returns NULL.", KEY_N_SERVERS);

/* read swap_endian for inifile */
pbuf = readKey(fp, KEY_SWAP_ENDIAN);
if (pbuf != NULL)
  swapend = atoi(pbuf);
else
  ErrorLog("Error reading %s. readKey returns NULL.", KEY_SWAP_ENDIAN);

*nCrates = 0;
```

```
  *nCards  = 0;
  *nServers = 0;
  index = 1;

  ASSERT(nservers == 0, "Found no servers in init file");

  for (i=0; i<nservers; i++) {

    /* read number of SERVER_SECTION  from file */
    pbuf = read_env(fp);

    sprintf(sectBuf,SERVER_SECTION,i+1);
    if ( (strcmp(pbuf, sectBuf)) != 0 )
      {
printf("Error: Invalid INI file, %s. No entry for %s.\n",
       INI_FILE, sectBuf);
len = strcmp(pbuf, sectBuf);
ErrorLog("strcmp(%s, %s) = %u\n", pbuf, sectBuf, len);
return kErrNoLibrary;
      }

    /* Get server name */
    pbuf = readKey(fp, KEY_SERVER_NAME);
    strcpy(gServerNames[i], pbuf);
    ASSERT(gServerNames[i] == NULL, "Couldn't get Server Name");

    /* Get number of crates */
    pbuf = readKey(fp, KEY_N_CRATES);
    ncrates = atoi(pbuf);
    for (j=0; j<ncrates; j++) {
```

```
      pbuf = read_env(fp);
      sprintf(sectBuf, CRATE_SECTION, i+1, j+1);
      ASSERT( (strcmp(pbuf, sectBuf)) != 0,
      "Couldn't read crate section");

      /* Get number of cards */
      pbuf = readKey(fp, KEY_N_CARDS);
      ncards = atoi(pbuf);
      if (ncards > 0) {
gGoodCrates[i][j] = 1;
if (buf != NULL) {
  buf[index++] = (unsigned short)i;
  buf[index++] = (unsigned short)j;
      }
gCrates[i][j] = malloc(sizeof(tCrate));

for (k=0; k<MAX_CARDS; k++)
  {
    if (k<ncards)
      gCrates[i][j]->Card[k] = malloc(256);
    else
      gCrates[i][j]->Card[k] = NULL;
  }

gCrates[i][j]->nCards = ncards;
for (k=0;k<ncards;k++) {
  sprintf(cardKey, KEY_CARD_NAME, k+1);
  pbuf = readKey(fp, cardKey);
  strcpy(gCrates[i][j]->Card[k],pbuf);
  ASSERT(pbuf==NULL,"Couldn't get card name");
```

```
  (*nCards)++;
}
(*nCrates)++;
      }
    }
    (*nServers)++;
  }
  if (buf != NULL)
    buf[0] = (unsigned short)((index - 1) / 2);

  ErrorLog("Found %d Servers, %d Crates, and %d Cards\n",*nServers,
   *nCrates,*nCards);

  return(0);
}



/*************************************************************************
* Initialization routine.  This routine calls read_ini_file(...) and returns
* the number of servers, crates and cards.
*************************************************************************/
long mit_caminit(short* buf)
{
  int nServers, nCrates, nCards;
  long ret;

  OpenErrorLog();
  ret = read_ini_file(buf, &nServers, &nCrates, &nCards);
  return ret;
}
```

```
/*************************************************************************
Routine to match xia's transfer function. Seems to be missing a way to report
the actual transfer length...
length is sent to error log if transfer is incomplete
*************************************************************************/
long mit_camxfr(short* cam_addr, short cam_func , long len,
short mode, unsigned short* buf)
{
  /* address; Input: address to access (CAMAC A)
   * cam_func; Input: cam_func number to address (CAMAC F)
   * len; Input: number of bytes to read or write
   * mode; Input: camac mode
   * buf; Input: data read or written *** data should be formatted as
   *                   unsigned shorts for !(mode & 24_BIT) and as unsigned
   *                   longs for (mode &24_BIT)
   */

  short iServ, iCrate, iCard, addr;
  char *routine;
  char *cardName;
  unsigned short iosb[4];
  int status, mem,  socket, i;
  long count, bytes;
  int vms = 0;
  char cmd[512];
  struct descrip ans_d = {0,0,{0,0,0,0,0,0,0,0},0};
#ifndef EXCLUDE_MDSLIB
  int returnLength = 1;
```

```
  int null = 0;
#endif

  iServ = cam_addr[HA_NMBR];
  iCrate = cam_addr[CRATE];
  iCard  = cam_addr [STA];
  addr = cam_addr[SUBADR];

  /* Execute a regular CAMAC command */
  /* verify crate is set up */
  if (gGoodCrates[cam_addr[HA_NMBR]][cam_addr[CRATE]] == 0 )
    {
      ErrorLog("Invalid Crate for address %d and crate #%d\n",
       cam_addr[HA_NMBR],cam_addr[CRATE]);
      return kErrInvalidCrate;   /* crate not setup */
    }

  /* establish a connection to our server... */
  if ((socket=gServerSocks[iServ]) == 0) {
#ifndef EXCLUDE_MDSLIB
    socket = MdsConnect(gServerNames[iServ]);
#else
    socket = ConnectToMds(gServerNames[iServ]);
#endif
    if (socket <= 0)
      {
ErrorLog("NoMDSConnection for socket %d\n",socket);
status = kErrNoMDSConnection;
return status;
      }
```

```
    gServerSocks[iServ] = socket;

#ifndef EXCLUDE_MDSLIB
    socket = MdsSetSocket(&socket);
#endif
    ErrorLog("MdsConnection available for socket %d\n",socket);
  }

  cardName = gCrates[iServ][iCrate]->Card[iCard];

  vms =  IsVMSServ(socket);
  mem = (mode & MODE_24_BIT) ? 24 : 16;   /* set 16 or 24 bit transfers*/
#ifdef  FORCE_MODE_16BIT
  mem = 16;
#endif
  bytes = (mem == 24) ? 4 : 2;  /* data transferred as USHORTS or ULONGS */
  count = len/bytes;

    if (!(cam_func & 8)) {   /*  0 <= f < 8    is read
      *  8 <= f < 16   is control
      *  16 <= f <= 32 is write
      *  f < 8 are reads f > 8 are writes
      */
      if( vms == 1) {

switch (mode & MODE_MASK) {
case MODE_STOPW:
  routine = "cam$stopw";
  break;
case MODE_SCAN:
```

```
  routine = "cam$qscanw";
  break;
case MODE_QSTOP:
  routine = "cam$qstopw";
  break;
case MODE_QREPEAT:
  routine = "cam$qrepw";
  break;
}
status = VMSCamMulti(socket, "cam$qstopw", cardName, addr, cam_func,
     count, (void *) buf, mem, iosb);
      }
      else /* window or *nix system */
{
  switch (mode & MODE_MASK)
    {
    case MODE_STOPW:
      routine = "Stopw";
      break;
    case MODE_SCAN:
      routine = "Qscanw";
      break;
    case MODE_QSTOP:
      routine = "Qstopw";
      break;
    case MODE_QREPEAT:
      routine = "Qrepw";
      break;
    }
  status = DoCamMulti(socket, routine, cardName, addr,
```

```
      cam_func, count, (void *) buf, mem, iosb);
}
    }
    else /* we are doing just single length thing... */
      {
if (vms == 1)
  status = VMSCamSingle(socket, "cam$piow", cardName, addr,
cam_func, 0, (void *) buf, 16, iosb);
else
  status = CamSingle(socket, "Piow", cardName, addr, cam_func,
     (void *) buf, 16, iosb);
      }

    if (!(status & 1))
      {
ErrorLog("Error sending camac transfer.  Status = %d", status);
return status;
      }

    /* Check camac status bits, X and Q */
    status = GetCamXandQ(socket);
    if (status == 1)
      {
ErrorLog("%s", "Status = CamX&Q\n");
return CAMXANDQ;
      }
    status = GetCamX(socket);
    if (status == 1)
    {
      ErrorLog("%s","Status = CamX");
```

```
      return CAMX;
    }
    else /* transfer was unsuccessful */
      {
status = kErrNotComplete;
bytes = GetCamBytCnt(socket);
ErrorLog("\nERROR! Transfer Incomplete. %08lX of %d bytes transferred",
 bytes, len);
return status;
      }
}

/*****************************************************************************
 * Function sends a question to MdsPlus server and returns an integer answer.
 *****************************************************************************/
long MdsQuery(SOCKET socket, char *question)
{
  struct descrip ans_d = {0, 0, {0,0,0,0,0,0,0,0}, 0};
  int null = 0;
  int returnLength = 1;
  int status;

#ifndef EXCLUDE_MDSLIB
  status = MdsValue(question,&ans_d,&null,&returnLength);
#else
  status = MdsValue(socket, question, &ans_d, 0);
#endif
  if ((status & 1) &&
      (ans_d.dtype == DTYPE_LONG) &&
      (ans_d.ptr != NULL))
```

```
    {
      memcpy(&status,ans_d.ptr,4);
      MdsIpFree(ans_d.ptr);
      ans_d.ptr = 0;
    }
  return status;
}

/*************************************************************************
 * Function request values of camac status bits X and Q from MdsPlus server.
 * Returns a value of 1 is X=1 and Q=1.
 *************************************************************************/
int GetCamXandQ(SOCKET socket)
{
  char *question = "CamXandQ()";
  long status;
  status =  MdsQuery(socket, question);
  return (int)status;
}

/*************************************************************************
 * Function request values of camac status bit X from MdsPlus server.  Returns
 * a value of 1 is X=1.
 *************************************************************************/
int GetCamX(SOCKET socket)
{
  char *question = "CamX()";
  long status;
  status =  MdsQuery(socket, question);
  return (int)status;
```

```
}

/*****************************************************************************
 * Function asks server for number of bytes transferred to or from camac module
 * in last operation. Returns the number of bytes transferred.
 *****************************************************************************/
long GetCamBytCnt(SOCKET socket)
{
  char *question = "CamBytCnt()";
  return MdsQuery(socket, question);
}

/*****************************************************************************
 * Returns 1 is MdsPlus server is running VMS operating system, 0 otherwise.
 *****************************************************************************/
int IsVMSServ(SOCKET socket)
{
  char *question = "VMS()";
  long status;

  status = MdsQuery(socket,question);
  if (status & 1)
    return 1;
  else
    return 0;
}

/*****************************************************************************
 * Function determines the number of 16 or 24 bit words to be transferred to
 * the camac module from the input variable, the number of bytes to be
```

```
 * transferred.
 ******************************************************************************/
int getNumWords(int nbytes, int mem)
{
  /* nbytes is number of bytes to be tranferred
   * mem is number of bits per transfer */
  int nWords;
  int temp;
#ifndef MESA2X
  int bytesPerWord = 2;
#else
  int bytesPerWord = (mem == 16) ? 2 : 4;
#endif

  nWords = nbytes/bytesPerWord;

  /* check to see if there is a leftover unsigned short */
  if  ((temp = nbytes % bytesPerWord) == 2)
    nWords++;

  return nWords;
}

/******************************************************************************
 * Writes data in hex format to output stream defined in ErrorLog((x),(y))
 * macro.
 ******************************************************************************/
void LogData(struct descrip data)
{
  int i,j,datasz;
```

```
#ifdef DEBUG
  /* Log the data */
  datasz=data.dims[0];
  if (datasz >=10)
    ErrorLog("Read %d words.\n", datasz);
  ErrorLog("%s"," Data = [ ");

  if (datasz >= 16)
    j = 8;
  else
    j = datasz;

  /* log all data or first 8 words of large blocks */
  for (i=0;i<j;i++)
    switch (data.dtype)
      {
      case DTYPE_LONG:
ErrorLog("%08lX ", *((long *)data.ptr + i));
break;
      case DTYPE_ULONG:
ErrorLog("%08lX ", *((unsigned long *)data.ptr + i));
break;
      case DTYPE_USHORT:
ErrorLog("%04hx ", *((unsigned short *)data.ptr + i));
break;
      }

  /* log last 8 words of data for large blocks */
  if (datasz >= 16)
```

```
    {
      ErrorLog("%s","...\n");
      for (i=(datasz-8);i<datasz;i++)
{
  switch (data.dtype)
    {
    case DTYPE_LONG:
      ErrorLog("%08lX ", *((long *)data.ptr + i));
      break;
    case DTYPE_ULONG:
      ErrorLog("%08lX ", *((unsigned long *)data.ptr + i));
      break;
    case DTYPE_USHORT:
      ErrorLog("%04hx ", *((unsigned short *)data.ptr + i));
      break;
    }
}
    }
  ErrorLog("%s","]\n");

#endif
}

/*------------------------------LINUX ROUTINES----------------------------*/

/***************************************************************************
 * From remcam libary, this routine performs a single length camac command.
 * Modified to use either MdsLib or MdsIpShr based on EXCLUDE_MDS_LIB macro.
***************************************************************************/
```

```
int CamSingle(int serverid, char *routine, char *name, int a, int f, void *data, int
{
  int status = kErrNoMDSConnection;
  int writeData;
  struct descrip data_d = {8,0,{0,0,0,0,0,0,0,0},0};
  struct descrip ans_d = {0,0,{0,0,0,0,0,0,0,0},0};
  char cmd[512];
  int null =0;
  int returnLength = 0;

  if (serverid)
  {
    writeData = (!(f &0x08)) && (f > 8);
    sprintf(cmd,"CamSingle('%s','%s',%d,%d,%s,%d,_iosb)",routine,name,
    a,f,(writeData) ? "_data=$" : "_data",mem);
    ErrorLog("CamSingle: F=%d, A=%d ,mem=%d ",f,a,mem);

    if (writeData)
      {
data_d.dtype = mem < 24 ? DTYPE_USHORT : DTYPE_ULONG;
data_d.ptr = data;
#ifndef EXCLUDE_MDSLIB
status = MdsValue(cmd,&data_d,&ans_d,&null,&returnLength);
#else
status = MdsValue(serverid,cmd,&data_d,&ans_d,0);
#endif
      }
    else
      LogData(data_d);
#ifndef EXCLUDE_MDSLIB
```

```
      status = MdsValue(cmd,&ans_d,&null,&returnLength);
#else
      status = MdsValue(serverid,cmd,&ans_d,0);
#endif

    if ((status & 1) &&
     (ans_d.dtype == DTYPE_LONG) &&
     (ans_d.ptr != NULL))
      {
memcpy(&status,ans_d.ptr,4);
MdsIpFree(ans_d.ptr);
ans_d.ptr = 0;
if (data && f < 8)
  getdata(serverid,data);
      }
  } /* end if serverid */
  return status;
}

/*****************************************************************************
 * From remcam library, this routine performs a read or write command to a
 * camac module.  Modified to use MdsLib or MdsIpShr.
*****************************************************************************/
int DoCamMulti(int serverid, char *routine, char *name, int a, int f,
       int count, void *data, int mem, unsigned short *iosb)
{
  int status = 0;
  int writeData;
#ifndef EXCLUDE_MDSLIB
  int null =0;
```

```
  int returnLength = 0;
#endif

  if (serverid)
    {
      struct descrip data_d = {8,1,{0,0,0,0,0,0,0,0},0};
      struct descrip ans_d = {0,0,{0,0,0,0,0,0,0,0},0};
      char cmd[512];
      writeData = (!(f &0x08)) && (f > 8);
      sprintf(cmd,"CamMulti('%s','%s',%d,%d,%d,%s,%d,_iosb)",routine,name,
      a,f,count,writeData ? "_data=$" : "_data",mem);

      ErrorLog("CamMulti F=%d, A=%d, mem=%d",f,a,mem);

      if (writeData)
{
  data_d.dtype = mem < 24 ? DTYPE_USHORT : DTYPE_ULONG;
  data_d.dims[0] = count;
  data_d.ptr = data;

  LogData(data_d);
#ifndef EXCLUDE_MDSLIB
  status = MdsValue(cmd,&data_d,&ans_d,&null,&returnLength);
#else
  status = MdsValue(serverid,cmd,&data_d,&ans_d,0);
#endif
}
      else /*read data */
{
#ifndef EXCLUDE_MDSLIB
```

```
  status = MdsValue(cmd,&ans_d,&null,&returnLength);
#else
  status = MdsValue(serverid, cmd,&ans_d,0);
#endif
}

      if (((status & 1)!=0) &&
  (ans_d.dtype == DTYPE_LONG) &&
  (ans_d.ptr != NULL))
{
  memcpy(&status,ans_d.ptr,4);
  MdsIpFree(ans_d.ptr);
  ans_d.ptr = 0;
  if (data && f < 8)
      getdata(serverid,data);
}
    }
  return status;
}

/*****************************************************************************
 * modified (for VMS) from mdsplus remcam library
 * This function gets data from the mdsPlus server.
 *****************************************************************************/
void getdata(int serverid, void *data)
/* Handel wants the data returned as unsigned shorts when mode = 4
 * MESA2X wants data returned as unsigned longs for mode = 5
 * so don't convert data. If 16 bit transfer is done, return
 * data will be unsigned short and if 24 bit transfer is done
 * return data will be unsigned long. */
```

```
{
  int status;
  struct descrip answer_d = {0,0,{0,0,0,0,0,0,0,0},0};
  unsigned short *pdata;
  int i, type;
#ifndef EXCLUDE_MDSLIB
  int null =0;
  int returnLength= 0;
#endif

  /* retrieve data from server */
#ifndef EXCLUDE_MDSLIB
  status = MdsValue("_data",&answer_d,&null,&returnLength);
#else
  status = MdsValue(serverid,"_data",&answer_d,0);
#endif

  /* LINUX and MACOSX MdsPlus libraries read data as Ushorts or Ulongs
   * while VMS MdsPlus libraries return data as Ushorts or signed longs */

  if (((status & 1) != 0) &&
      ((answer_d.dtype == DTYPE_USHORT) ||
       (answer_d.dtype == DTYPE_ULONG) ||
       (answer_d.dtype == DTYPE_LONG)) &&
      (answer_d.ptr != NULL))
    {
      memcpy(data,answer_d.ptr,((answer_d.dtype == DTYPE_USHORT) ? 2 : 4) *
     answer_d.dims[0]);
      LogData(answer_d);
    }
```

```
  else
    ErrorLog("\nERROR getting data.  Status = %d\n", status);

  if (answer_d.ptr != NULL)
    MdsIpFree(answer_d.ptr);
}


/*---------------------------------VMS ROUTINES-------------------------------*/

/*****************************************************************************
 * This routine performs a single length camac command.Used for control
 * functions are are writes, 16 bit.
*****************************************************************************/
int VMSCamSingle(int mdsSock, char *routine, char *name, short address,
   short function, long count, void *data, int mem,
 unsigned short *iosb)
{
  int status = 0;
  char cmd[512];
  struct descrip ans_d = {0x08,'\0',{0,0,0,0,0,0,0},0};
  struct descrip data_d = {0x08,'\1',{0,0,0,0,0,0,0},0};
  int null = 0;
  int returnLength = 0;

  data_d.dtype = DTYPE_USHORT;
  data_d.dims[0] = count;
  data_d.ptr = data;

  ErrorLog("F(%d)*A(%d)\tbytes: %ld\t\t",function,address,count);
```

```
  sprintf(cmd,"_iosb=zero(4,0wu),_data=$");
#ifndef EXCLUDE_MDSLIB
  status = MdsValue(cmd,&data_d,&ans_d,&null,&returnLength);
#else
  status = MdsValue(mdsSock,cmd,&data_d,&ans_d,NULL);
#endif
  if (ans_d.ptr != NULL)
    MdsIpFree(ans_d.ptr);
  ans_d.ptr = 0;
  sprintf(cmd,
  "CamShr->%s('%s',long(%d),long(%d),ref(_data),long(%d),ref(_iosb))",
  routine,name,address,function,mem);

#ifndef EXCLUDE_MDSLIB
  status = MdsValue(cmd,&ans_d,&null,&returnLength);
#else
  status = MdsValue(mdsSock,cmd,&ans_d,NULL);
#endif

  if ((status == 1)&&
      (ans_d.ptr != NULL) &&
      (ans_d.dtype == DTYPE_ULONG))
    {
      memcpy(&status, ans_d.ptr, 4);
      MdsIpFree(ans_d.ptr);
      if (!(status & 1))
ErrorLog("MDS camac error: status = %d",status);
    }
  else
    ErrorLog("Error excuting command, %s. Status = %d.\n", cmd, status);
```

```
  return status;
}

/*****************************************************************************
 * This routine performs the io call to read or write data using the CamShr
 * function.  Adapted from old code to run XerXes.
*****************************************************************************/
int VMSCamMulti(int mdsSock, char *routine, char *name, short address,
short function, long count, void *data, int mem,
unsigned short *iosb)
/* int *camChan;     Input: pointer to camac crate to access     */
/* unsigned int *address;   Input: address to access (CAMAC A)         */
/* unsigned int *function;  Input: function number to access (CAMAC F) */
/* unsigned int *count;     Input: words to read or write */
/* void *data;     I/O:  data read or written        */
{
  /*
    IDL> print,mdsvalue("_iosb=zero(4,0wu)")
    0       0       0       0
    IDL> print,mdsvalue("_iosb")
    0       0       0       0
    IDL> print,mdsvalue(" _status = CamShr->Cam$Piow('ldx_dxp_1',_a,_f,\
    ref(_data),_mem,ref(_iosb))")
    1
    IDL> print,mdsvalue(" _status = CamShr->Cam$qstopw('ldx_dxp_1',_a,_f,\
    ref(_c),ref(_data),_mem,ref(_iosb))")
  */

  int status = 0;
```

```
  int i = 0;
  int len;
  unsigned short *pdata = NULL;
  struct descrip data_d = {8,1,{0,0,0,0,0,0,0,0},0};
  struct descrip ans_d = {0,0,{0,0,0,0,0,0,0,0},0};
  char cmd[512];
  int null = 0;
  int returnLength = 0;
  int writeData = (!(function &0x08)) && (function > 8);

  /* initialize data using mds */
  if (writeData)
    {
      /* initialize data descriptor */
      if (mem==16)
{
#ifdef MESA2X

  /* reformat data for 16 bit transfers - remove excess zeros */
  pdata = malloc(count*2);
  for (i=0;i<count;i++)
    pdata[i] = ((unsigned short*) data)[2 * i];

#else
  pdata = data;
#endif
}
      data_d.dtype = mem < 24 ? DTYPE_USHORT : DTYPE_ULONG;
      data_d.dims[0] = count;
      if (mem==16)
```

```
data_d.ptr = pdata;
      else
data_d.ptr = data;

      /* initialize variables on MdsServer */
      sprintf(cmd,"_iosb=zero(4,0wu),_data=$,_c=size(_data)");
#ifndef EXCLUDE_MDSLIB
      status = MdsValue(cmd,&data_d,&ans_d,&null,&returnLength);
#else
      status = MdsValue(mdsSock,cmd,&data_d,&ans_d,NULL);
#endif
    }
  else /* read data */
    {
      /* initialize variables on MdsServer */
      if (mem==24)
sprintf(cmd,"_iosb=zero(4,0wu),_data=zero(%ld,0u),_c=long(%ld)",
count,count);
      else
sprintf(cmd, "_iosb=zero(4,0wu),_data=zero(%ld,0wu),_c=long(%ld)",
count,count);
#ifndef EXCLUDE_MDSLIB
      status = MdsValue(cmd,&ans_d,&null,&returnLength);
#else
      status = MdsValue(mdsSock,cmd,&ans_d,NULL);
#endif
    }
  if (ans_d.ptr != NULL)
    MdsIpFree(ans_d.ptr);
  ans_d.ptr=0;
```

```
  /* execute CamShr->routine command */
  sprintf(cmd,"CamShr->%s('%s',%d,%d,ref(_c),ref(_data),%d,ref(_iosb))",
  routine,name,address,function,mem);
#ifndef EXCLUDE_MDSLIB
  status = MdsValue(cmd,&ans_d,&null,&returnLength);
#else
  status = MdsValue(mdsSock,cmd,&ans_d,NULL);
#endif

  if ((status == 1)&&
      (ans_d.ptr != NULL) &&
      (ans_d.dtype == DTYPE_ULONG))
    {
      memcpy(&status, ans_d.ptr, 4);
      MdsIpFree(ans_d.ptr);
      if (status & 1)
{
  if (!writeData)
    getdata(mdsSock,(void *)data);
}
      else
ErrorLog("MDS camac error: status = %d",status);
    }
  else
    ErrorLog("Error excuting command, %s. Status = %d.\n", cmd, status);

  if (pdata != NULL)
    free(pdata);
```

```
  return status;

}
/*-----------------------READ INI FILE ROUTINES----------------------------*/

/*****************************************************************************
 * This routine reads charaters from an input file until the target character
 * is reached.  The output is a pointer to the string of characters upto but
 * not including the target.  The target is discarded.
*****************************************************************************/
char* read_to_char(FILE* fp, char target)
{
  static char buf[256];
  static char *pbuf;
  char c;
  int i=0;

  pbuf = &buf[0];
  while ( (c=getc(fp)) != target)
    {
      if (c == EOF)
{
  ungetc(c, fp);
  return NULL;
}
      buf[i] = c;
      i++;
    }

  buf[i] = '\0';
```

```
  return pbuf;
}

/*****************************************************************************
 * This routine reads a string beteen square brackets and returns a pointer
 * to the string.
*****************************************************************************/
char* read_env(FILE* fp)
{
  static char *pbuf;

  /* read string between brackets */
  pbuf = read_to_char(fp, '[');
  pbuf = read_to_char(fp, ']');

  return pbuf;
}

/*****************************************************************************
 * This routine reads a word preceded by white space and followed by an
 * equals sign.
*****************************************************************************/
char* read_value(FILE* fp)
{
  static char *pbuf;
  char c;

  /* remove any white space before value */
  while (isspace(c = getc(fp)))
```

```
    {
      if (c == EOF)
{
  ungetc(c, fp);
  return NULL;
}
    }

  /* unget last character, first character of value */
  ungetc(c,fp);

  /* read word */
  pbuf = read_to_char(fp, '=');
  return pbuf;
}

/*****************************************************************************
 * function reads a keyword out of the inifile.
 *****************************************************************************/
char* readKey(FILE* fp, char* key)
{
  char* buf;
  int len = 0;

  buf = read_value(fp);
  if ( (strcmp(buf,key)) != 0 )
    {
      ErrorLog("Error: Invalid INI file, %s. No entry for %s.\n",
       INI_FILE, key);
      len = strcmp(buf, key);
```

```
      ErrorLog("Output of strcmp is %d. Program reads _%s_, but wants _%s_\n",
       len, buf, key);
      return NULL;
    }
  else
    {
      buf = read_word(fp);
#ifdef DEBUG
      ErrorLog("Function readKey outputs: %s\n", buf);
#endif
      return(buf);
    }
}

char* read_word(FILE *fp)
{
  static char buf[256];
  static char *pbuf;
  char c;
  int i=0;

  pbuf = &buf[0];

  /* remove white spaces before word */
  while (isspace(c = getc(fp)))
    {
      if (c == EOF)
{
  ungetc(c, fp);
  return NULL;
```

```
}
    }

  /* last character is not a space, so it is the first
   * character of the word */
  buf [i] = c;

  /* read the rest of the  word */
  while(!isspace(c=getc(fp)))
    {
      i++;
      buf[i] = c;
    }

  /* unget last white space */
  ungetc(c, fp);

  buf[i+1] = '\0';
  return pbuf;
}
```

Bibliography

[1]

[2] Spear detector datasheet.

[3] A. C. Boxer, J. Kesner, D. T. Garnier, A. K. Hansen, and M. E. Mauel. Interferometry for ldx? In *presented at 45th Annual Meeting of the American Physical Society Division of Plasma Physics*, Albuquerque, NM, October 2003.

[4] J. L. Ellsworth, J. Kesner, D. T. Garnier, M. E. Mauel A. K. Hansen, and S. Zweben. X-ray diagnostics for ldx. In *presented at 45th Annual Meeting of the American Physical Society Division of Plasma Physics*, Albuquerque, NM, October 2003.

[5] D. T. Garnier, J. Kesner, and M. E. Mauel. Ecrh on the levitated dipole experiment. In *presented at 13th Topical Conference on Applications of Radio Frequency Power to Plasmas*, Annapolis, MD, April 1999.

[6] I. H. Hutchinson. Chapter 5.3: Radiation from electron-ion encounters. In *Principles of Plasma Diagnostics*, pages 186–214, Cambridge, 2002. Cambridge University Press.

[7] I. Karim, J. Kesner, D. T. Garnier, and M. E. Mauel A. K. Hansen. Magnetic diagnostics in ldx. In *presented at 45th Annual Meeting of the American Physical Society Division of Plasma Physics*, Albuquerque, NM, October 2003.

[8] J. Kesner, L. Bromberg, M. Mauel, D. T. Garnier, and J.M. Dawson. The dipole fusion confinement concept: A white paper for the fusion community. Technical Report PSFC/RR-98-5, MIT Plasma Science and Fusion Center, April 1998.

[9] J. Kesner and D. T. Garnier. Convective cell formation in levitated dipole. *Phys. Plasmas*, 7:2733, 2000.

[10] J. Kesner, D.T. Garnier, A. Hansen, M. Mauel, and L. Bromberg. D-d fusion in a dipole. *Need to find out*, 2003.

[11] E. E. Ortiz, J. Kesner, D. T. Garnier, O. Grulke, A. K. Hansen, and M. E. Mauel. Diagnostic setup for spacial and temporal measurements of plasma fluctuations using electric probes on ldx. In *presented at 45th Annual Meeting of the American Physical Society Division of Plasma Physics*, Albuquerque, NM, October 2003.

[12] S. von Goeler, S. Jones, R. Kaita, S. Bernabei, W. Davis, H. Fishman, G. Gettelfinger, D. Ignat, F. Paoletti, G. Petravich, F. Rimini, P. Roney, J. Stevens, and W. Stodiek. Camera for imaging hard x rays from suprathermal electrons during lower hybrid current drive on pbx-m. 65:1621, May 1994.

[13] X-Ray Instrumentation Associates, Newark, CA. *Users Manual: Digital X-Ray Processor Model DXP2X*, a edition, June 2002.