A Self-Triggered Readout for a Time Projection Chamber

by

Andrew Thompson Werner

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Physics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2004

Author . . . .
Department of Physics
December 9, 2003

Certified by . . . .
Peter H. Fisher
Professor, Department of Physics
Thesis Supervisor

Accepted by . . . .
David Pritchard
Senior Thesis Coordinator, Department of Physics

A Self-Triggered Readout for a Time Projection Chamber

by

Andrew Thompson Werner

Submitted to the Department of Physics
on December 9, 2003, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Physics

Abstract

A self-triggering readout for a time projection chamber (TPC) is presented, with applications to novel forms of data acquisition for high energy physics application. The construction and initial testing of the readout electronics are described, as well as the readout implementation in a prototype drift chamber. Tracking and range information for 800 alpha particle events is compared with theory as a preliminary confirmation of correct readout operation.

Thesis Supervisor: Peter H. Fisher
Title: Professor, Department of Physics

Acknowledgments

There are a great many of people who helped bring this project to where it is today. I owe a great deal of thanks to Profs. Ulrich Becker and Kate Scholberg for their help and expertise at all stages of this project. Thanks also to the students in Building 44: Ben Monreal, Reyco Henning, Bilge Demirköz, Gianpaolo Carosi, Shi Yue, and Blake Stacey. Fred Cote was very helpful in fabricating the aluminum collimator used in conjunction with the alpha particle source, and Mike Grossman provided invaluable assistance constructing and reworking the physical chamber. Thanks to Sherilyn Harrison for the design of the DD-1 multilayer PC board, and to Mark Henebry of Argo Transdata Corporation for supervising its construction.

Additionally, much thanks is due to Prof. Joseph Paradiso and Josh Lifton of the Responsive Environments group of the MIT Media Laboratory. Without their countless hours of work on the Pushpin Computing platform, as well as their consistent willingness to help at any time, this thesis would not have been possible.

Thanks also to Bernard Wadsworth, formerly of the LNS electronics department and currently retired. Bernie was an immense help with the electronic design of the DD-1 readout, taking time out of his own schedule to provide absolutely priceless assistance, and teaching me a great deal at the very same time.

Tremendous thanks is due to my tireless collaborator, Andrew Campanella. Andy has shared in many successes and frustrations with me over the course of this project, and throughout our association, it has been an unqualified pleasure to work with him.

Finally, I owe the greatest debt of gratitude to Prof. Peter Fisher, for making this thesis possible. Being able to participate in this project and collaborate with Peter has been one of the best experiences of my life, and is undoubtedly the defining experience of my time at MIT. Words cannot adequately express how much this opportunity has meant to me.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The demands of modern high energy physics necessitate experiments of great complexity and cost. Those experiments that make use of particle accelerators are emblematic of this fact. The detectors required to acquire data from these machines are very large and complicated arrays of sensors, which obtain their information, as a whole, by looking at aggregated information from their constituent components.

In general, data is acquired in such experiments by a series of successive reductions of data. Of the many interactions that are to be expected in a high energy collision, perhaps only a few are of interest to the experimenter. As a result, modern detectors are designed to look quickly at subsets of the data associated with a collision and, based on this, make quick decisions about the relevance of the collision in question. The hardware and software systems responsible for this decision making are known collectively as *triggers*. Triggers are categorized by "level"; each successive level presents an increasingly restrictive set of criteria that an electronic signal has to meet in order to be passed on to the next level trigger. A signal that passes all levels of triggers is then recorded as an *event* corresponding to appropriate physics. A typical example would be the trigger for the L3 experiment at the LEP collider at CERN, described in [3]. At the collision point, e^+e^- interactions occurred at a rate of approximately 40 kHz. Physically interesting events were filtered from this data at a rate of ~ 1 Hz That data that is then recorded can be then reduced into the various kinds of data that is appropriate to analyses and then processed offline.

While technological advancements have permitted vast increases in the capability of these detection systems, the organization of the data acquisition has remained fundamentally unchanged since the inception of these large scale systems. In particular, all computational tasks are centralized and global, meaning that they occur with full knowledge of all parameters of the equipment for which they process, either on the detector or subdetector level. Moreover, the data acquisition occurs synchronously, and is hence the rate at which data can be acquired is limited by the slowest step in the process. This raises an interesting question: is it possible to achieve greater efficiencies, in cost or computation, by (1) distributing the centralized processing power to the domain of the individual sensor, and (2) dispensing with the requirement that the data acquisition and processing be globally synchronous?

The motivation for the material discussed in this thesis is to build a functional detector that accomplishes meaningful physics tasks through distributed processing. In essence, the aim of this work is to lay a foundation, using a Time Projection Chamber as an initial testbed, for a paradigm of data acquisition that is amenable to decentralized computing topologies.

1.2 The Time Projection Chamber

A Time Projection Chamber (TPC) is a type of detector that provides tracking information about charged particles that traverse its fiducial volume. Such information is generally in the form of coordinates (x_i, y_i, z_i) at a resolution of approximatly 100 points per meter of track length. TPC operation is depicted in Figure 1-1, and is described in greater detail in [7]. TPC designs, while varied, share a variety of elements, including the following:

- A *drift volume*, consisting of a noble gas (with a quencher) and an (ideally) constant electric field, called the *drift field*. The noble gas is chosen to have

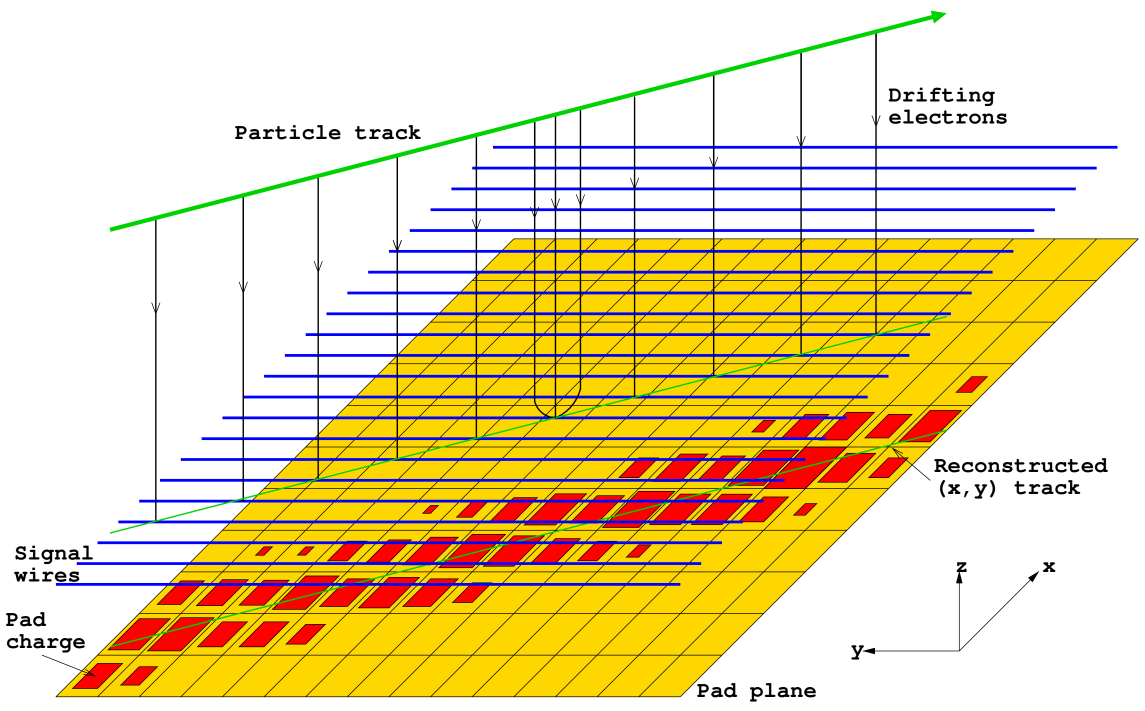


Figure 1-1: Graphical representation of TPC operation. A passing particle produces ionization in the detector volume, which drifts in the z-direction under the influence of a high electric field. Amplication occurs at the sense wires, causing an induced charge on the cathode pads. Image from [14].

a high atomic number Z so that particles traveling through the volume lose sufficient energy per unit length to produce a signal.

- A *grid*, which screens the electrons in the region of the sense (or anode) wires from the ions in the drift volume. This is helpful in ensuring a uniform drift field.

- *Sense wires*, which are held at high potential and amplify the primary electron signal through the gas avalanche mechanism.

- *Cathode pads* are located near to the sense wires and inductively pick up the signal from the sense wire avalanche.

In general, both the pads and sense wires are connected to electronic instrumentation. The spatial distribution of charge in the cathode pad plane gives information about the track projection in the xy-plane, while the drifting electrons' arrival times at the sense wires give information about the z-coordinate of the incident track.

TPCs have found use in many modern particle physics experi.ments. The ALEPH and DELPHI detectors at the LEP collider at CERN both utilized a TPC [9, 4], as does the STAR TPC at RHIC. [15]

1.3 Amorphous Computing

The challenge of rethinking the conventions of data acquisition in particle detectors is itself analogous to the challenge posed by the problem of *amorphous computing.* [5] Amorphous computing has as its goal the achievement of computationally useful tasks by distributed networks of computers (hereafter *nodes*), the constituents of which satisfy the following criteria:

1. The nodes are all identical, in hardware and software.
2. Each node can communicate only with its closest neighbors.
3. Only a majority of nodes are required to function.

In particular, for our prototype detector, called the DemoDetector-1 or DD-1, we do not require that our nodes:

- operate synchronously, or
- know their specific environmental parameters.

Additional work on this topic can be found in the work of Butera, [2] whose paradigm of "Paintable Computing" provided the direct inspiration for this project, and of Lifton et.al., [8] whose hardware implementation of Butera's concept provided a physical platform around which the detector described in this thesis could be based.

Our discussion will proceed as follows. Chapter 2 will introduce the basic operational principles of a TPC as specifically relevant to this thesis. Chapter 3 will discuss the hardware and software development for the DD-1 signal readout. We follow with results and analysis in Chapter 4, and conclusions in Chapter 5.

Chapter 2

Principles of Operation

A TPC is capable of creating a three dimensional track of a charged particle that passes through its fiducial volume. The information about the track manifests itself in two major components. First, there is a two dimensional distribution of charge on the *cathode plane* of the TPC, giving the x- and y -coordinates of the particle's path. Second, the z-coordinate is given by the time distribution of the drifting electrons' arrival at the sense wires, or, correspondingly, the generation of the signal on the cathode pads.

In the DD-1 detector, only the x- and y-coordinates are measured by the readout electronics. In the course of the construction and testing of the chamber, however, both pad and wire signals were examined, and so we will discuss the generation of both.

2.1 Data Acquisition Methodology

2.1.1 Terminology

The key point of departure between DD-1 and other drift chamber designs is in the data acquisition methodology. In DD-1, event detection is a purely local phenomenon, organized around the *node*. A DD-1 node is shown in Figures 2.1.1 and 2.1.1. The solder side of the board, having the six rectangular pads, is itself the cathode plane

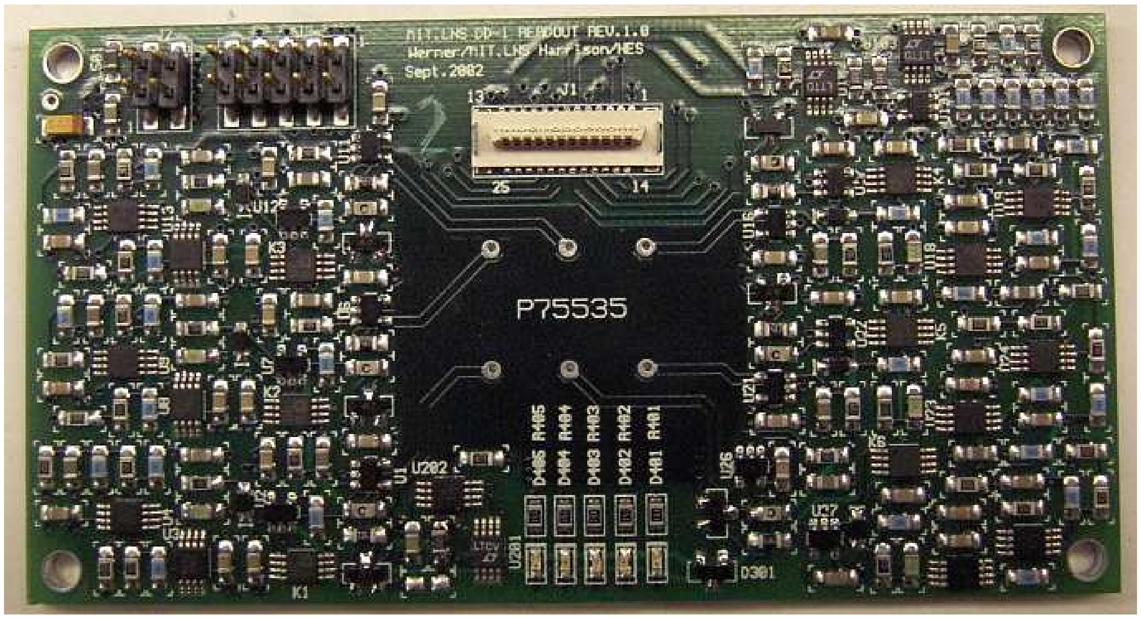


Figure 2-1: Component side of the six-channel DD-1 readout board. The 25 pin CONAN connector to the Pushpin daughterboard is visible in the center. The power is supplied through the four pin Molex connector in the upper left hand corner, and the JTAG interface is through the ten pen Molex connector nearby.

of the chamber. The pad signals are amplified and processed by the circuitry on the component side of the board, which includes an 8051 microcontroller based daughterboard.

The convention used for the DD-1 coordinate system is shown in Figure 2-3. A row of three pads, all centered at the same x-coordinate, is known as an *xrow*. The quantities Δx and Δy referred to in the Figure are 11 mm and 7 mm, respectively on the boards used for this thesis.

2.1.2 Track Construction

The physical event most important to us in this thesis is simply a charged particle passing through the DD-1 volume. The signal resulting from the drift of the ionization electrons to the sense wires is then read by the DD-1 nodes in the chamber. Details of the signal generation are discussed in Section 2.2.

An individual node acquires track information as follows.

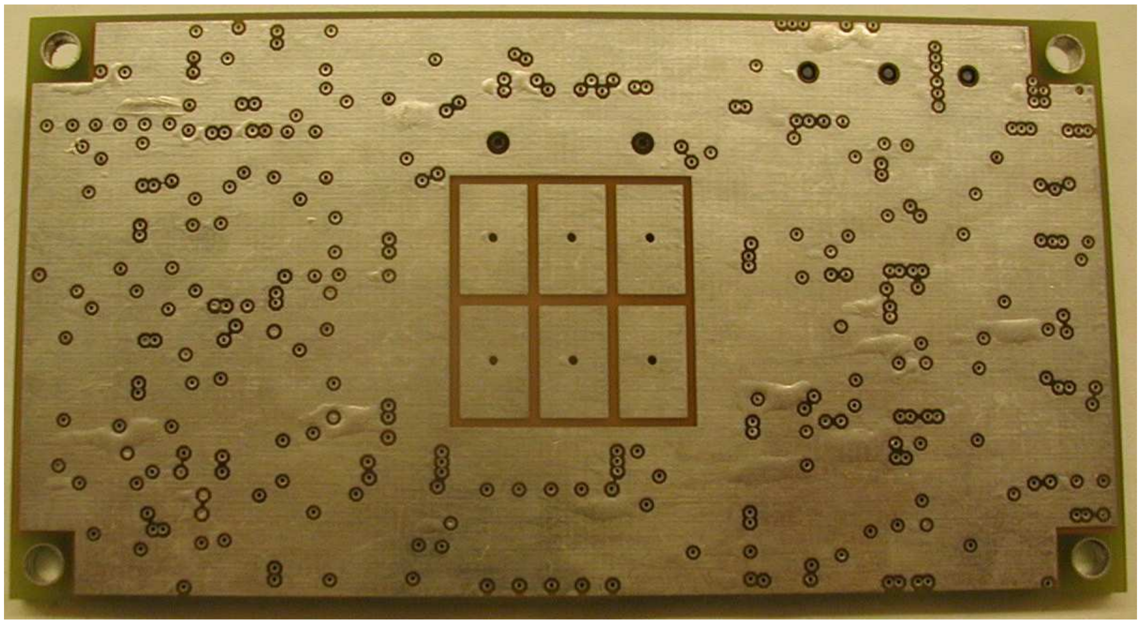


Figure 2-2: Solder side of the DD-1 readout board. The cathode pads are clearly visible.

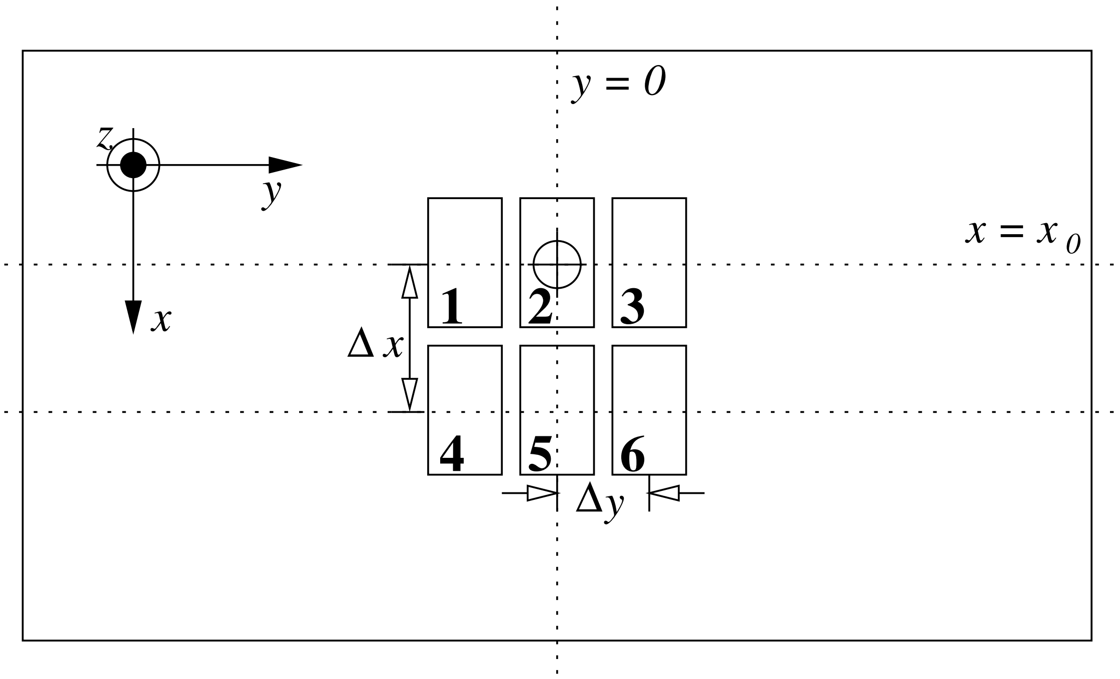


Figure 2-3: Schematic of DD-1 readout board solder side, with coordinate conventions notated. The pad numbering convention is denoted with the boldface numerals.

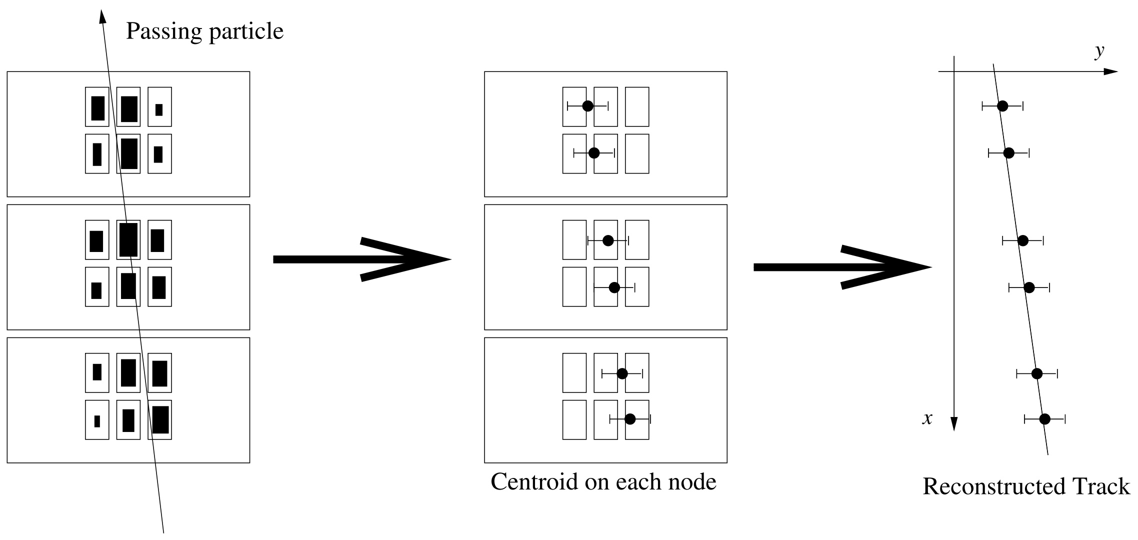


Figure 2-4: Schematic representation of reconstruction of particle track over several DD-1 nodes.

1. A particle passing overhead induces a signal in the node pads. If the sum of the signals on all pads is of sufficient amplitude (defined in software), we say that a *trigger* has just occurred on the board.

2. The microcontroller reads the peak charge induced on each pad due to the passing particle.

3. The software on the microcontroller (or external computer, in the case of the implementation of DD-1 considered in this thesis) determines, for each xrow, the y position of the particle track at the center x_0 of the xrow.

This process gives two points (x_i, y_i) of the particle track for each node, with errors on the y-coordinate determined through the fit procedure. Combining this data over the ensemble of nodes allows for a determination of a particle track through the entire fiducial volume of the chamber. Knowledge of information for more nodes will allow for a more precise determination of the track. See Figure 2-4.

2.2 Signal Generation

2.2.1 Track Ionization

A particle that passes through a TPC leaves its signature in the form of a trail of electron-ion ion pairs along its path. The number of pairs that we can expect to produce per unit of track length is most frequently understood in terms of the incident particle energy loss per unit length, dE/dx. This is given to a good approximation by the *Bethe-Bloch* formula: [7]

$$-\frac{dE}{dx} = 2\pi N_A r_e^2 m_e c^2 \rho \frac{Z}{A} z^2 \beta^2 \left[\ln \left(\frac{2 m_e \gamma^2 v^2 W_{max}}{I^2} \right) - 2\beta^2 \right]. \qquad (2.1)$$

Here, N_A is Avogadro's number, r_e is the classical electron radius, and m_e is the electron mass. Z, A, and ρ are the atomic number, atomic weight, and density, respectively, of the absorbing material. I is the mean excitation potential of the absorbing material, v is the velocity of the incident particle, and W_{max} is the maximum energy transfer possible in a single collision.

The Bethe-Bloch formula is only valid, however, from highly relativistic particles down to particles for which $\beta \simeq 0.1$. In the case of 5.5 MeV α particles, we have $\beta \simeq 0.05$. The majority of ionization in the DD-1 chamber is due to argon gas, which is kept at a partial pressure of approximately 0.9 atmospheres. Assuming the validity of the Bethe-Bloch formula down to the $\beta \simeq 0.05$ regime, we calculate $-dE/dx = 1.8$ MeV/cm due to the presence of argon in the chamber alone. This indicates to us that not far into the track of the incident particle, the Bethe-Bloch will be of vanishingly little use in calculating the expected distance that alpha particles might travel in the gas. Consequently, the use of computer calculations will be essential in our analysis, as there is no analytical means of characterizing dE/dx over the entire expected length of our alpha particle tracks. [7]

2.2.2 The Gas Avalanche

The amount of ionization produced per unit track length is generally small relative to what can be detected with conventional instrumentation. As a result, drift chambers generally employ a means of proportional amplification to increase the magnitude of the electron signal.

The mechanism by which this is accomplished is known as avalanche multiplication. The fields in the TPC drift volume are such that electrons will drift in a direction perpendicular to the cathode plane, toward the sense wire plane. In the immediate vicinity of the sense wires, the electric field is strong, becoming a better approximation to the $\sim 1/r$ $\mathbf{E}$-field of an isolated infinite wire as the distance to a particular wire decreases.

Electrons that have energy greater than or equal to the first ionization potential E_i of a drift gas can create electron-ion pairs. The drifting electrons also undergo elastic scattering with gas molecules. The mean distance traversed by an electron before scattering is called the *mean free path* for scattering. Drifting electrons in the vicinity of a sense wire also gain energy due to high electrostatic fields. For a given electric field, there is a length that an electron must traverse before it has energy sufficient to create an electron-ion pair. This is known as the *mean free path for ionization*, and its inverse, α, is known as the *first Townshend coefficient*. The avalanche begins when the mean free path for ionization and scattering become comparable. [13]

In regions of sufficiently high electric field, the effects of ionization by secondary collision are high enough to account for an amplification of the electron signal. Parameterizing α by the spatial coordinate x, it is possible to determine the factor by which one expects an electron signal to be amplified. This is given by: [13]

$$M = \exp\left(\int_{x_1}^{x_2} \alpha(x) dx\right). \tag{2.2}$$

2.2.3 The Wire Signal

The electron avalanche in the vicinity of a wire produces electron-ion pairs. The electrostatics in the avalanche region cause the electrons to drift towards the sense wire, while the ions have a tendency to drift either ground plane—the grid or the cathode.

The grid-wire-cathode system may be regarded as a capacitor, and changes in the energy of the capacitor are realized as observable signals. The motion of a single charge in the capacitor reduces the electric energy E of the system by an amount [1]

$$\Delta E = \int_{x_1}^{x_2} q\mathbf{E} \cdot d\mathbf{r} = q(\Phi_1 - \Phi_2), \tag{2.3}$$

where Φ_1 and Φ_2 denote the electrostatic potential at points x_1 and x_2, respectively. The distance traveled by the electrons in the avalanche is typically much less than that traversed by the ion, and so the motion of the ions contributes correspondingly more to the signal observed on the wires. Because the avalanche occurs in very close proximity to the sense wire, we may approximate, for every ion in the avalanche, $\Phi_1 - \Phi_2 \simeq V$, where V is the voltage on the sense wire referenced to ground. The change in capacitor energy as a function of time is then given by [1]

$$\Delta E = qVF(t), \tag{2.4}$$

where $F(t)$ is a function that rises monotonically from zero to one.

A typical scheme for observing the signal on a sense wire is shown in Figure 2-5. In the limit where R_2C_2 and R_1C_1 are large with respect to the risetime of the ion signal, the sense wire itself acts effectively as a voltage source, and we have [1]

$$\Delta V(t) = \frac{q}{C_1}F(t), \tag{2.5}$$

where $\Delta V(t)$ is the time-dependent voltage excursion of the sense wire from its steady state value. For the observational setup in Figure 2-5, typical peak observed $\Delta V(t)$ values were on the order of 10 to 20 mV, with a rise time on the order of 50 ns and

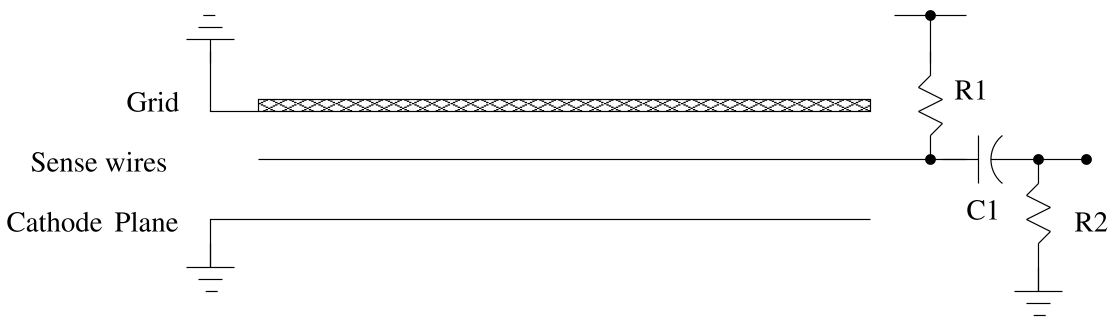


Figure 2-5: Typical connection to observe the sense wire signal in a drift chamber. In DD-1, $R_1 = 7.5\text{M}\Omega$, $R_2 = 100\text{k}\Omega$, $C_1 \sim 1\text{pF}$, and $C_2 = 1\text{nF}$. The resulting signal was plugged directly into a high impedence oscilloscope. Adapted from [1].

a fall time on the order of 1 μs.

2.2.4 The Pad Signal

The component of the avalanche signal that is actually used for determining track coordinates is produced on the aforementioned cathode pads. The pad signal is primarily the result of the production and subsequent drift of charged ions in the sense wire region immediately above the pads.

We define the normalized *pad response function* $P(y)$ as the amount of charge induced on a TPC pad centered at $(0, y)$ due to an infinitely long particle track in the $\hat{\mathbf{x}}$ direction at $y = 0$, with the convention that $P(0) = 1$. To derive this, we model the avalanches as point charges in space, and use the method of images to determine the surface charge induced on a ground plane a specified distance away from the sense wires as a function of position, [1]. The pad response function may then be determined by simple integration.

In actuality, the avalanche does not occur instantaneously; it is characterized by rising and falling time constants. Since the time constant for its rise is (by design) much quicker than that for its fall, the charge induced on the pad is taken to be the peak charge over time.

In the vicinity of its peak, the pad response function is well approximated by a Gaussian. We use this feature to simplify the process of locating the mean value of the charge distribution in our analysis.

Chapter 3

Implementation

3.1 Chamber Design

3.1.1 Physical Design

The DD-1 Chamber is a clear acrylic chamber, mounted on an aluminum stand. The drift cage, which ensures a uniform electric field over a large volume, is mounted on a solid aluminum backplane with nylon hardware. Dimensions are given in Figure 3-1

The chamber contains feedthroughs for the high voltage supplies that supply the voltages for the sense wire and for maintaining the drift field, for the JTAG interface to the DD-1 readout board, for the DC power required for signal readout, and for reading the signal off of a single sense wire. The high voltage distribution scheme is given in Figure 3-2, while the signal wire readout scheme was illustrated in Figure 2-5. The 7.5 MΩ resistors connecting each sense wire to the V_{an} rail ensure electrical isolation of each wire with respect to the others. This is essential to ensure that signals are properly induced on the pads.

3.1.2 Chamber Operation

The chamber was tested extensively prior to acquisition of the data sets used in this thesis. The gas used in the chamber during these tests and during normal chamber operation was P10 (Ar:CH_4 90:10), which was chosen for its high Z, high gain at

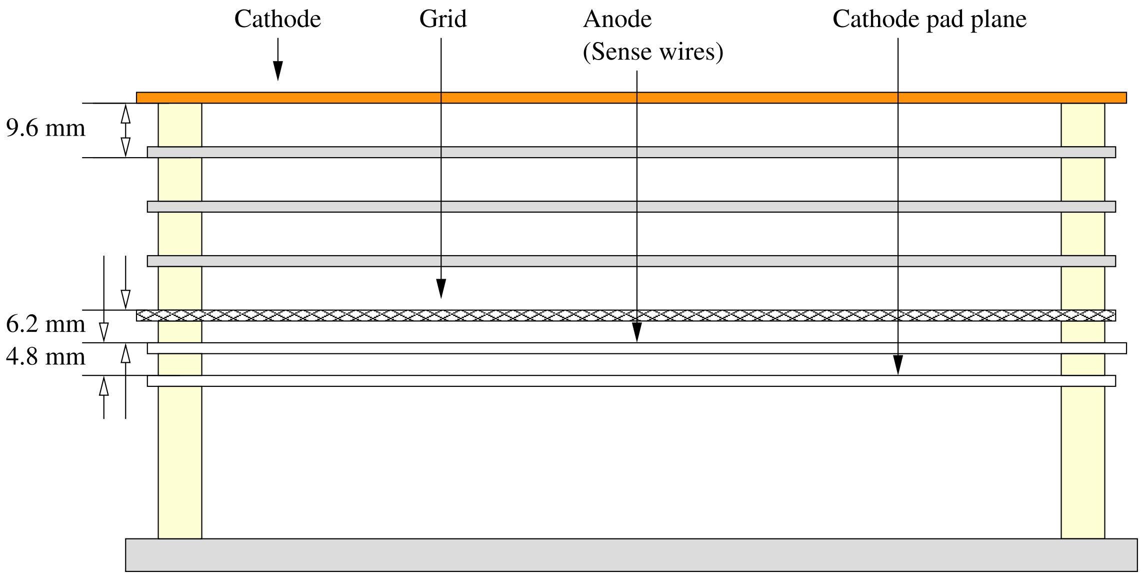


Figure 3-1: Schematic of the drift cage, sense wires, and cathode plane on the DD-1 detector, with dimensions. The drift region has an extent (in the z-direction) of approximately 3.9 cm.

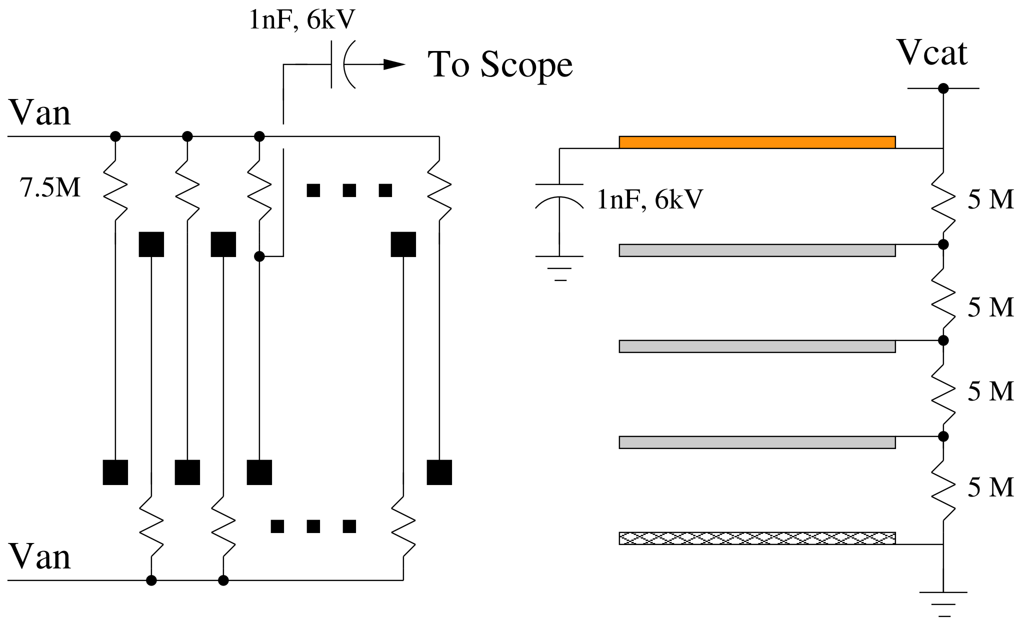


Figure 3-2: Schematic of the high voltage disctribution for the anode and cathode of the chamber.

relatively low sense wire voltages, and negligible electron attachment.

Early experimental runs were characterized to noise due to corona discharge between the sense wires and the readout boards. At very high anode voltages (~ 2.6 kV) the corona was visible; at lower anode voltages it made itself apparent through apparently ohmic behavior visible on the high voltage power supply and its internal voltage/current monitor.

The effect of the corona was minimized through the following procedures:

- Copper surfaces in the vicinity of high electric fields were cleaned with alcohol.
- The screening grid was replaced with a flatter grid, with tension kept in the grid mesh through the use of epoxy.
- The DD-1 readout board was replaced with one having a more nearly flat solder (pad plane) side.

Through these methods, stable operation was achieved for sense wire voltages of 2.2 kV.

The cathode voltage was kept at -600 V, corresponding to a drift field of roughly 150 V/cm in the center of the fiducial volume of the detector.

3.2 Signal Readout

The DD-1 pad readout circuit is a single circuit board, with six pads and a ground plane comprising the solder side of the board. The readout circuit is designed to operate off the same 3 volt power supply as the 8051 processor to which it sends its relevant information. The two main functions of the readout are (1) to determine, for each pad, the peak charge accumulated on that pad due to an incident particle, and (2) to generate a trigger signal for the 8051 daughterboard that indicates that a particle has passed above the node.

The basic block diagram of the circuit is given in Figure 3-3. Further details of the circuit can be found in Appendix A.

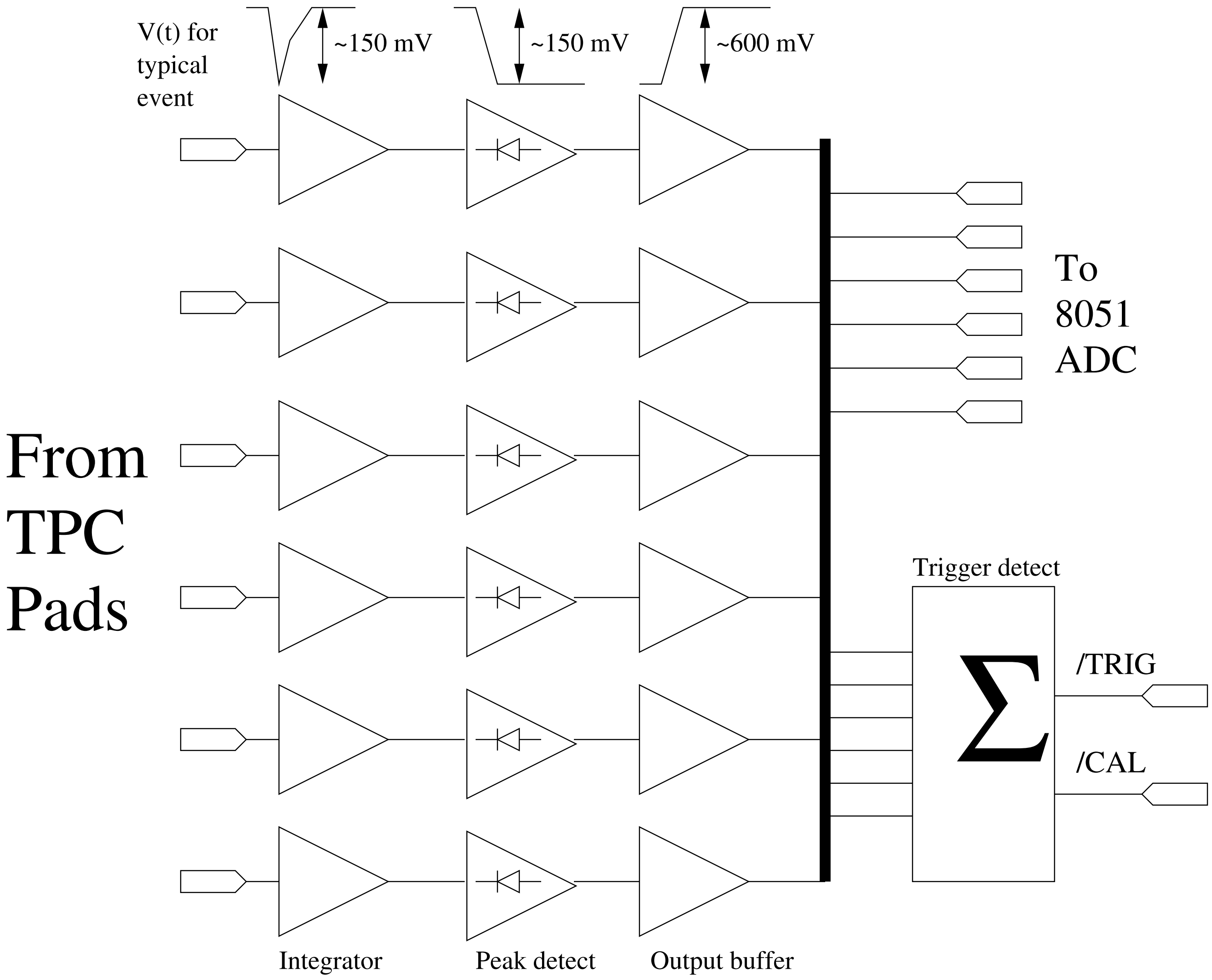


Figure 3-3: Block diagram of the DD-1 readout. Typical signal shapes at the end of each indicated stage are shown.

3.3 Digitization and Analysis

3.3.1 Creation of the Digital Signal

Digitization of the analog signal is accomplished by a Pushpin processing module [8], which itself consists of a Cygnal 8051F016 microcontroller [11] with limited support circuitry and connections to the main readout board. The microcontroller used in this configuration has 8 input channels which can be configured to perform analog to digital conversion. The initial DD-1 nodal configuration makes use of six of these channels, which are connected to the termini of the individual pad readout chains.

As of the writing of this thesis, only one of the two available interrupts (/TRIG) was implemented in software. As suggested by its name, a falling edge on /TRIG is read as a signal for the Pushpin to perform an ADC conversion on each of its input channels in succession.

Individual triggers are logged in software as C `struct`s:

```
typedef struct {
     unsigned long eventTicks;
     unsigned int eventClocks;
     unsigned char padValues[6];
} Event;
```

The `padValues` array contains the highest order 8 bits of the ADC conversion result for the associated pad, indexed by pad number. Combined, `eventTicks` and `eventClocks` count the number of 8051 Timer3 counts since system startup as a 48 bit unsigned integer. The high order byte is `eventTicks` and the low order is `eventClocks`.

3.3.2 Analysis

The built in JTAG interface to the Cygnal 8051F000 series contains a provision for making a numerical dump of the contents of XDATA memory to a personal computer. This was accomplished with the Cygnal Integrated Development Environment (IDE) software. [11, 12] A description of the process by which data is acquired from the Pushpin is given in Appendix C.

Event data was analyzed in Matlab. Data for aggregated events was typically kept in two variables. The first was a vector of "times," the number of ticks elapsed on the Pushpin Timer3 clock since system startup. This vector was used in the early stages of data analysis for investigations into the time correlations of suspicious events, which disappeared in the later analysis. The second is a $6 \times n$ matrix of pad values.

Each readout channel has a built in offset, or pedestal value. This value is determined in software by holding the RESET line high for an amount of time sufficient to zero the peak detect and hold stage, delaying to allow the circuit to stabilize, and then performing an analog read on the given channel. These pedestal values are then subtracted from the pad ADC readings to give a digital value which is, in principle, proportional to the charge deposited on the pad to within 1/2 LSB.

The analysis involves fitting a Gaussian to the ADC values for each xrow for each trigger, as an approximation for the true pad response. The model was of the form

$$P(y) = A \exp\left[\left(\frac{y-b}{8\text{ mm}}\right)^2\right], \tag{3.1}$$

where A and b are the fitted parameters, and the number 8 in the denominator of the exponential was determined by a graphical comparison with a a pad response function numerically calculated from chamber parameters. Throughout the analysis, distances are given in millimeters. The quantity b is then taken to be the y-position of the particle over the center of the xrow for which the fit is performed.

A comparison of the calculated pad response for the DD-1 chamber geometry and the Gaussian approximation is given in Figure 4-3.

Chapter 4

Data and Analysis

4.1 Summary of Data

For the purposes of testing the DD-1 chamber and readout, we used ^{241}Am as a source of 5.5 MeV α-particles. The source was collimated and placed in the chamber such that particles would have to travel less than 2 cm from the source before being directly over the cathode pads. A schematic of the setup, with a detail of the collimator, is shown in Figure 4-1. In the parlance of DD-1, xrow 2, consisting of pads 4, 5, and 6, was nearer the source for this particular run.

A total of 800 board triggers were recorded. A histogram of pad data for the events is shown in Figure 4-2.

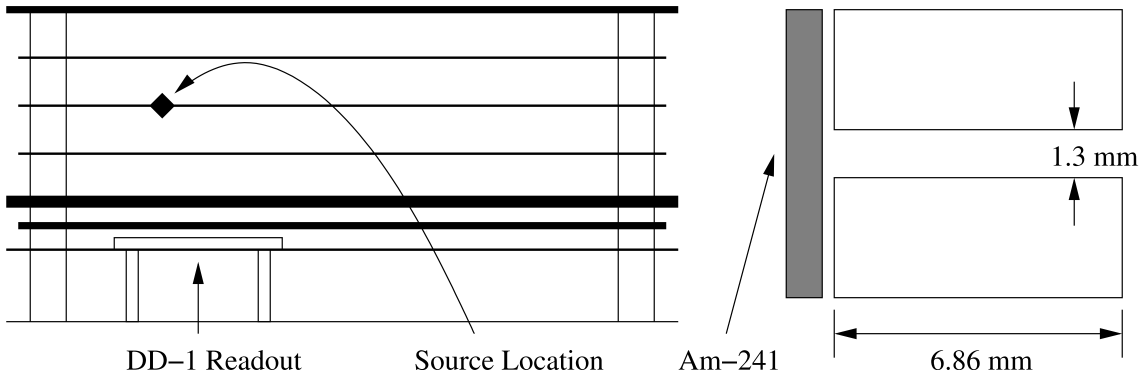


Figure 4-1: Setup for single node alpha particle events. Dimensions of the Al collimator are shown at right.

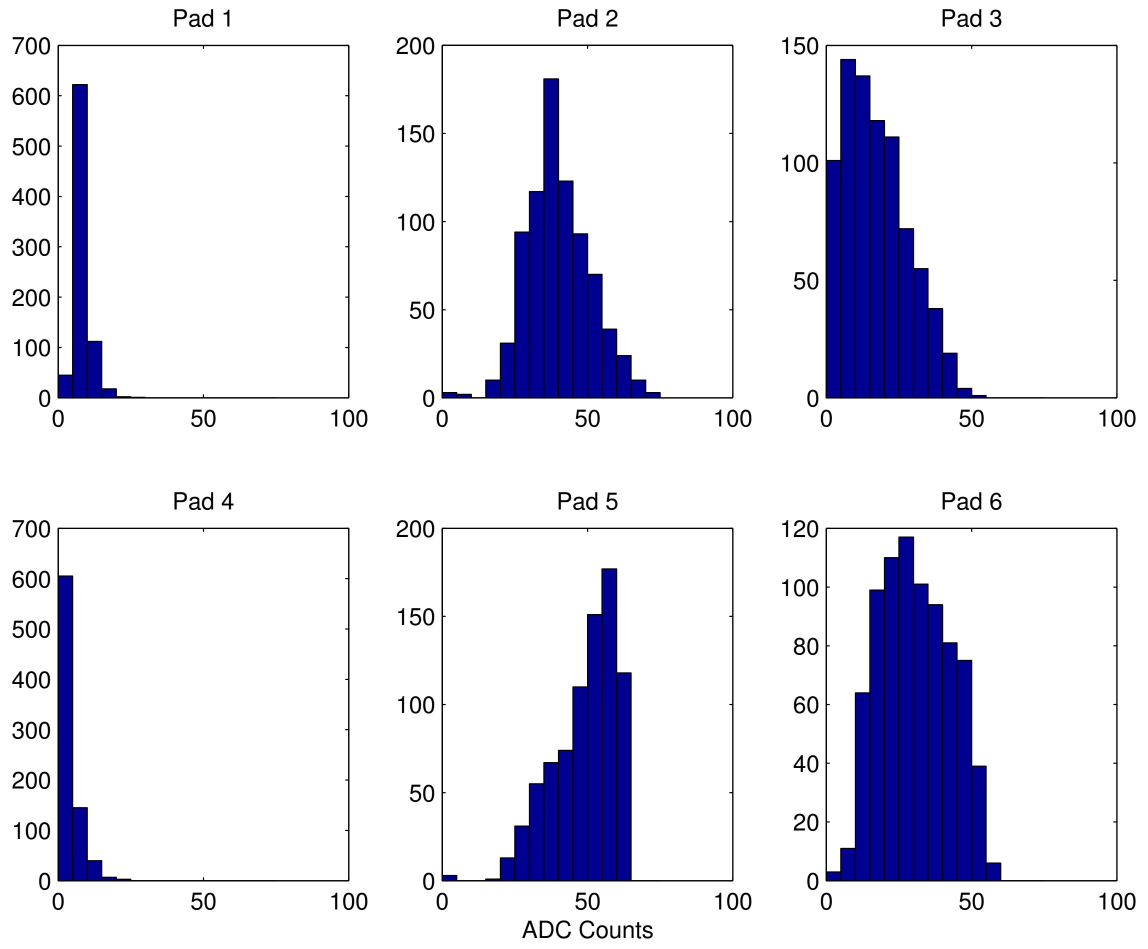


Figure 4-2: Histograms of ADC readings for 800 events, 3 September 2003, after pedestal subtraction.

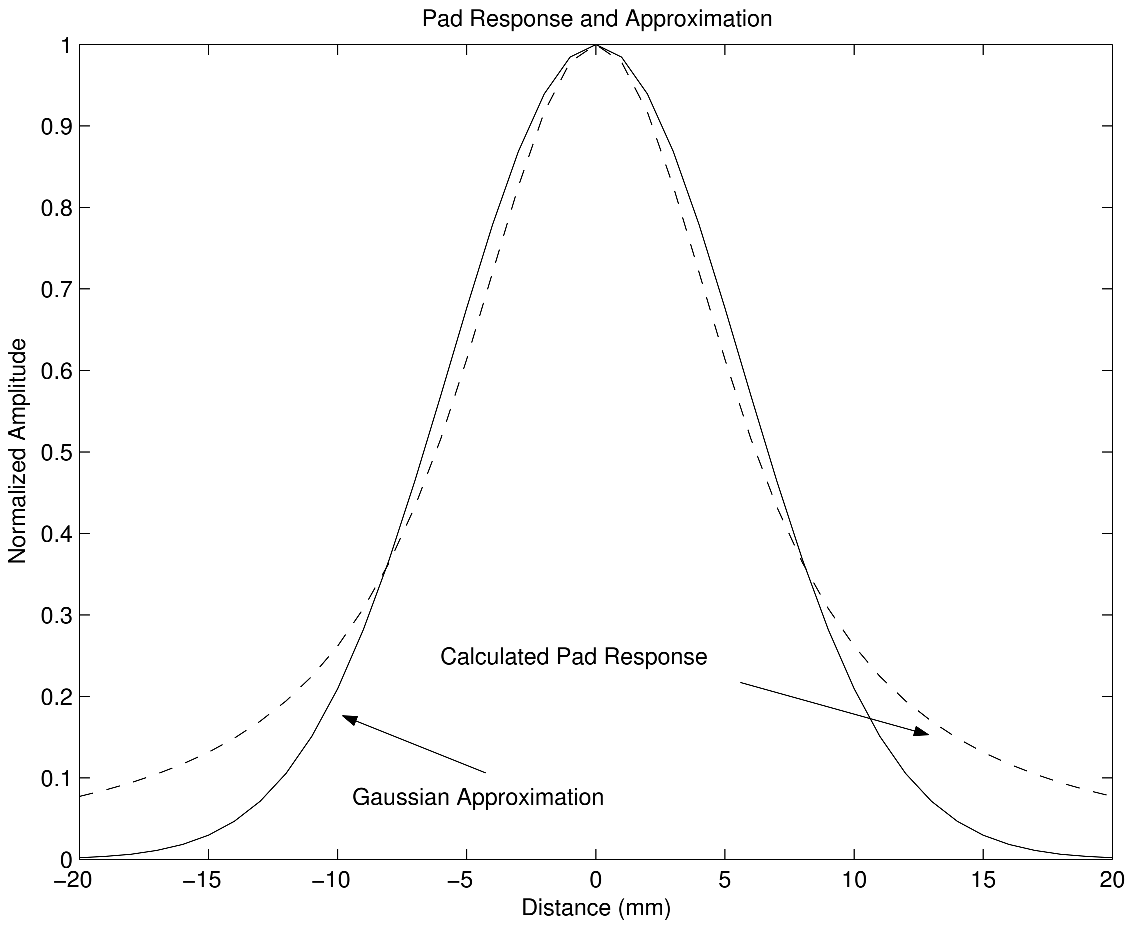


Figure 4-3: Comparison between calculated pad response function and Gaussian approximation with standard deviation 5.7 mm.

4.2 Results

4.2.1 Coordinate Measurements

Coordinate calculation was accomplished through a χ^2 fit of a Gaussian approximation to a calculated pad response function. A comparison of the calculated pad response, based on chamber parameters, and the Gaussian approximation used in the fitting model is shown in Figure 4-3. The distance between the centers of the pads is 7 mm, indicating that the Gaussian approximation should hold reasonably well over a three pad row. A sample fit to pad data is shown in Figure 4-4.

A histogram showing the computed centroid locations for each xrow for the 800

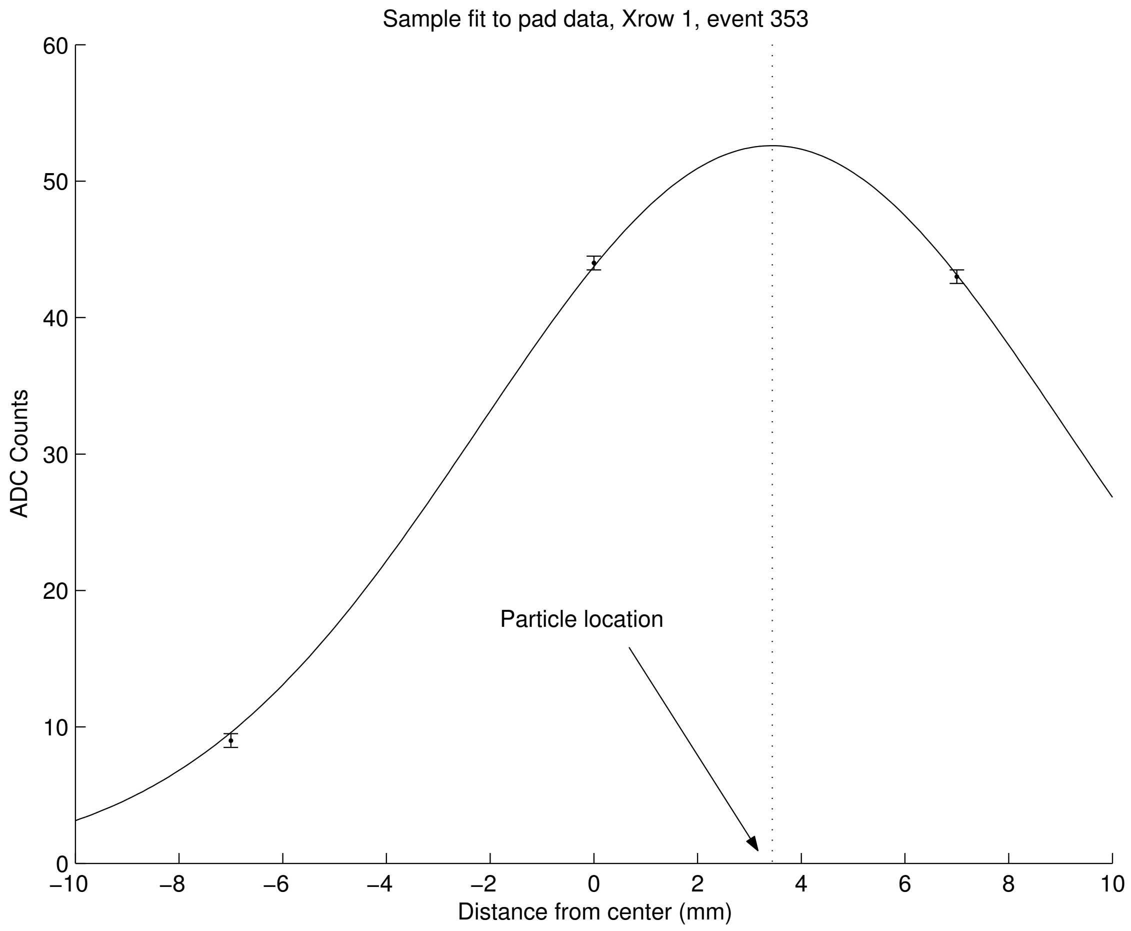


Figure 4-4: Example fit to pad data. Errorbars correspond to 1/2 LSB on the 8051 ADC.

event run depicted in Figure 4-2 is given in Figure 4-5. Below each histogram is the expected distribution of alpha particles, calculated with a Monte Carlo simulation. See Appendix B.

Xrow 1 exhibits good agreement with the predicted spread of the detected particles, with a slight rightward bias. Xrow 2, the nearer row, shows a greater spread than xrow 1 and does not fit with the Monte Carlo data in this regard. The peak, however, is centered in roughly the correct location, with a rightward bias similar to that exhibited by xrow 1.

Correlation between centroid locations in both xrows for individual events is demonstrated in Figure 4-6. The rightward bias of the xrow 2 coordinate is easily evident; however, the correlation between coordinates in each row is qualitatively what would be expected for our setup, particularly for xrow 2 coordinates nearer zero.

4.2.2 Energy Measurements

In producing primary pairs of electrons, a passing α-particle loses energy. The number of particles produced in a given length of track should then be proportional to the amount of energy lost by the particle in said length. Assuming proportional amplification, the same should then hold for the amount of charge induced on a row of cathode pads.

The amplitude of the Gaussian fitted to the charge deposition on a row of pads is, in principle, proportional to the energy loss of the passing particle on the short length of track directly above the row. Histograms showing this data for the set under examination are given in Figure 4-7. An event-by-event scatter plot of the data is show in Figure 4-8. No correlation between amplitudes in each xrow appears to be readily evident.

The differences between the distributions of energies observed on xrow 1 and xrow 2 appears to be indicative of α-particles coming to a stop over xrow 1. (Recall that xrow 1 is farther from the source than xrow 2.) However, it is difficult to correlate the number of ADC counts on xrow 1 with the distance traversed by an α-particle with any precision. The dE/dx of stopping α-particles is characterized by an increase

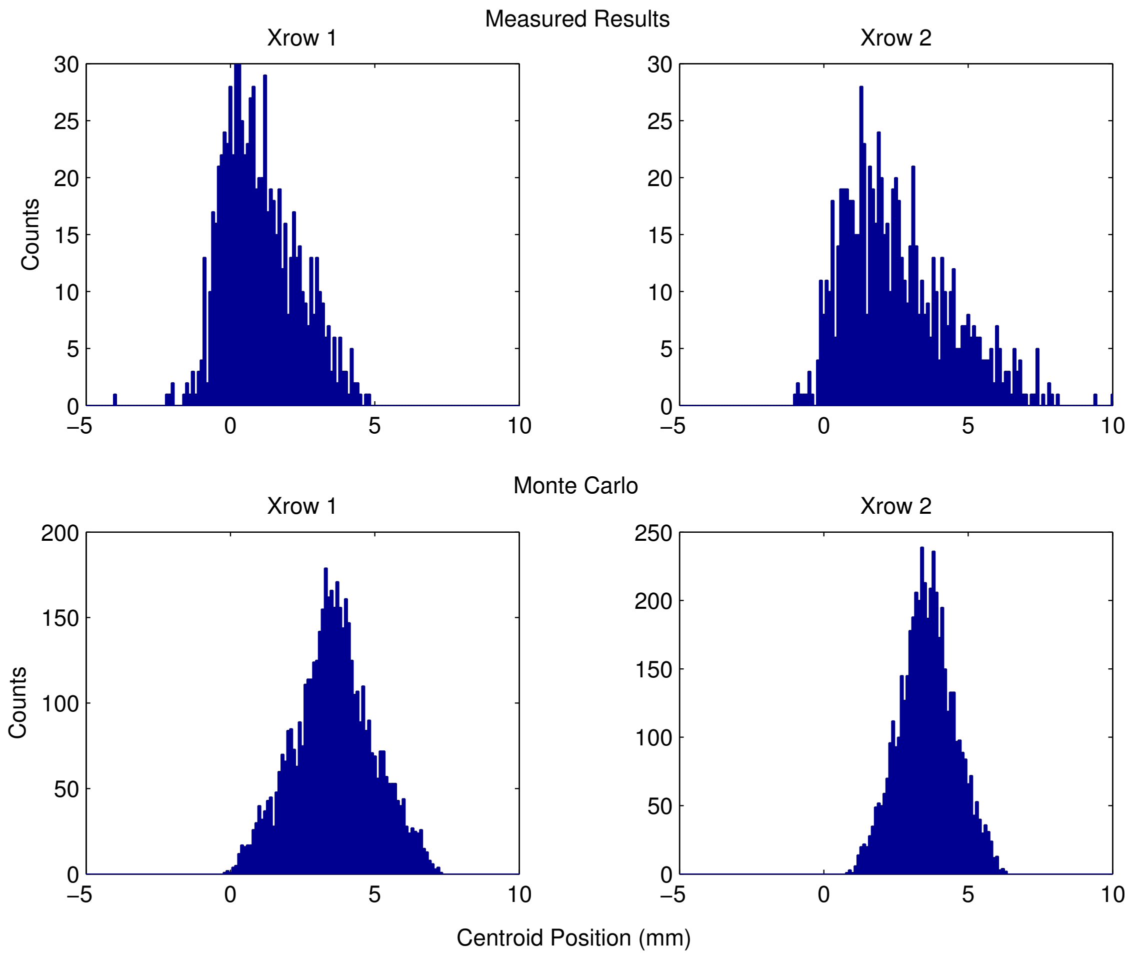


Figure 4-5: Fitted centroids of charge distributions shown for both xrows, with comparison to Monte Carlo simulation.

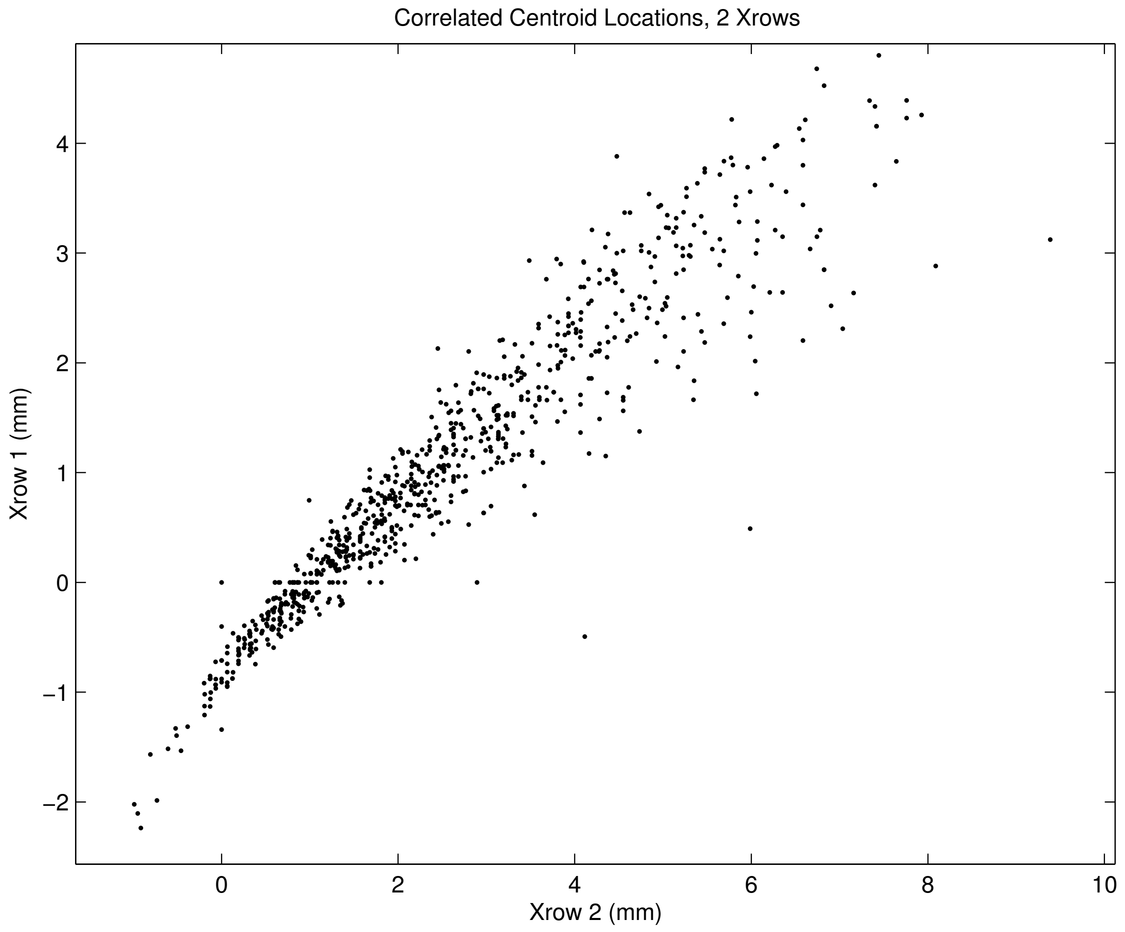


Figure 4-6: Scatter plot of coordinate measurements for individual events, both xrows.

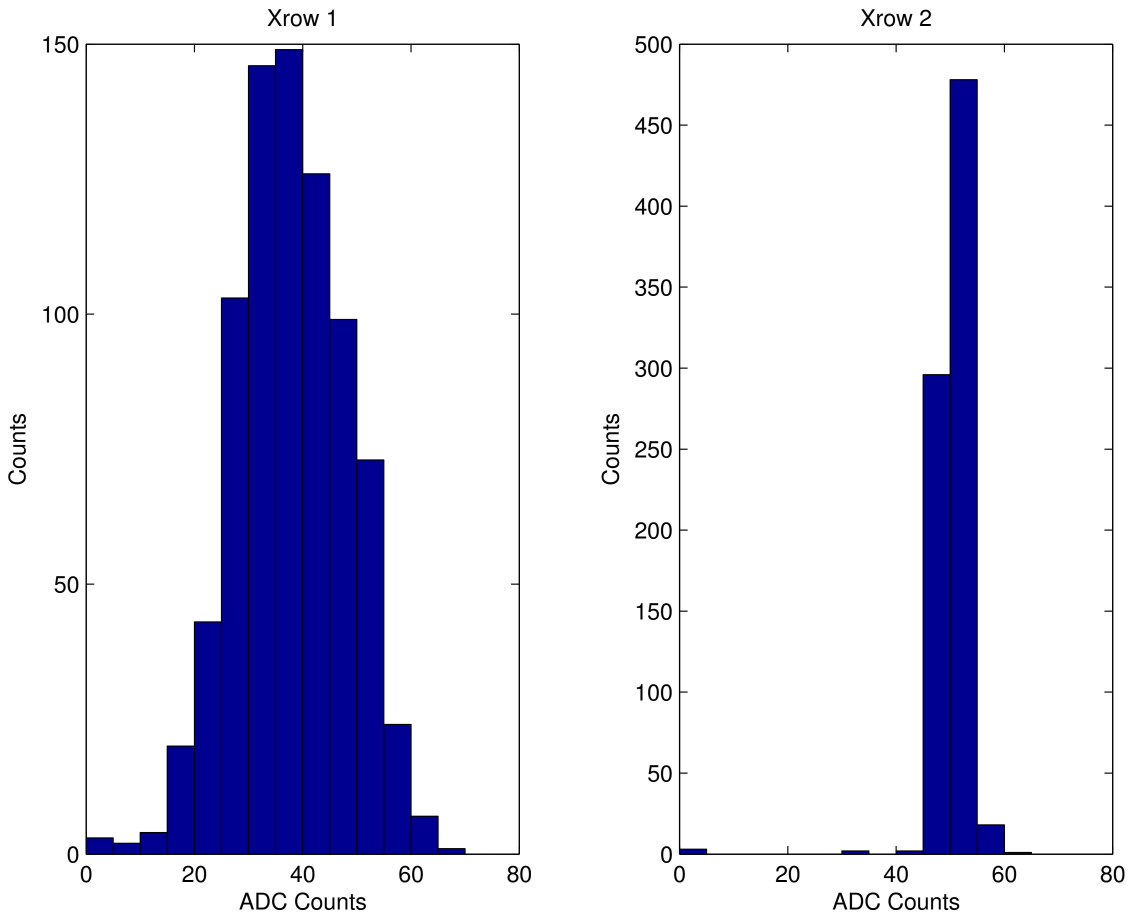


Figure 4-7: Histogram of charge profile amplitudes for both xrows.

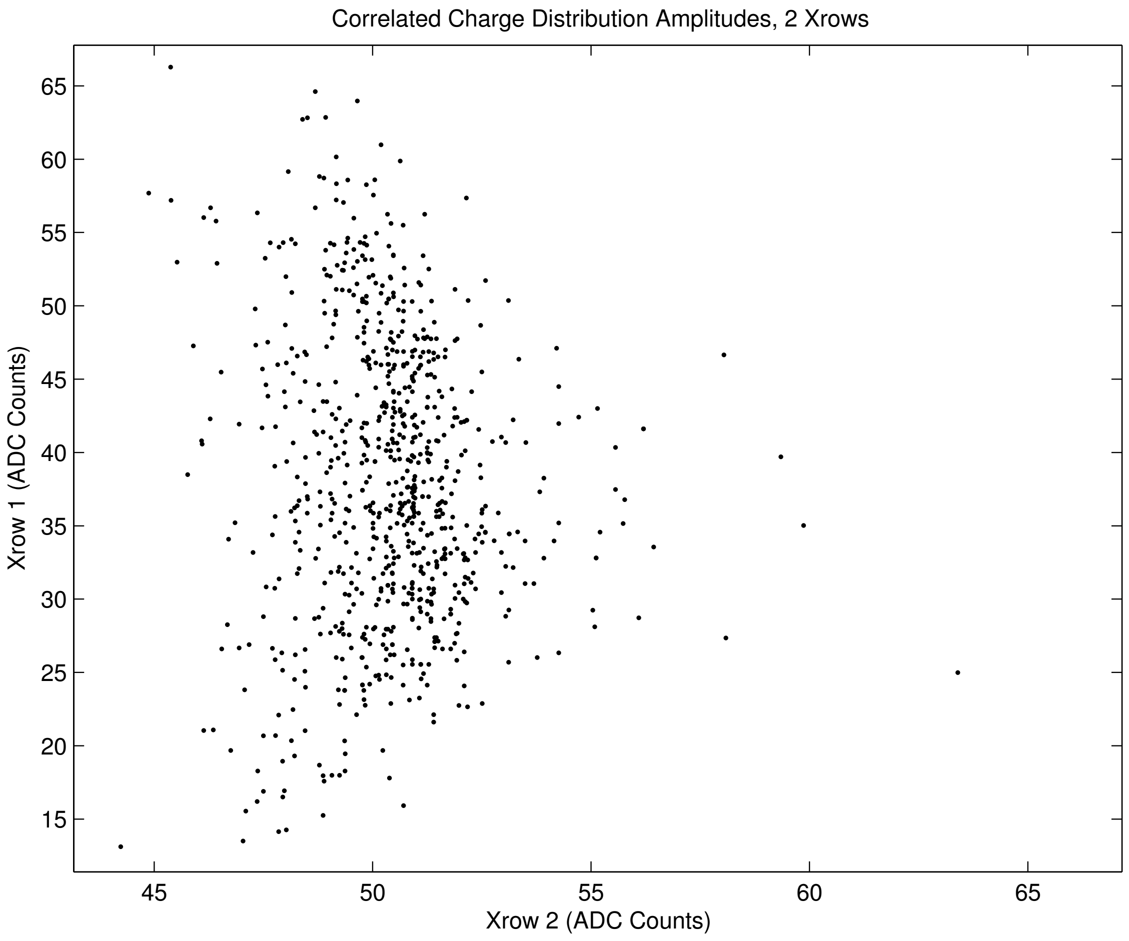


Figure 4-8: Scatter plot of charge profile amplitudes for individual events, both xrows.

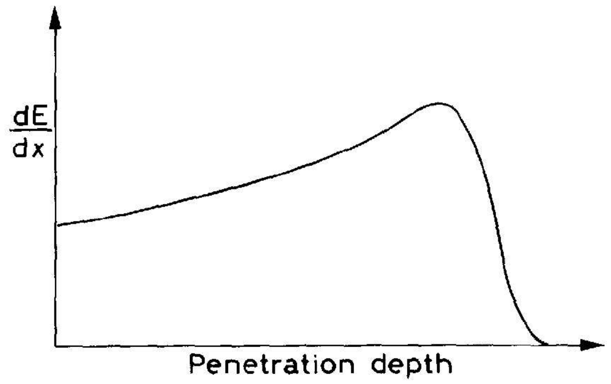


Figure 4-9: Example of a Bragg curve, showing the increase and subsequent decrease of dE/dx of an ionized particle as it stops in some medium.

as the particle slows down, followed by a sharp decrease as the particle captures electrons. This is represented schematically in Figure 4-9; such curves are known as *Bragg curves.* [7]

Because of the analytical difficulties with Bragg curves, we will hypothesize that on the average, the incident α-particles stop at the central x-coordinate of xrow 1. This corresponds to a distance of 3.40 cm from the source in the x-direction; this assertion seems to be roughly supported by the symmetry of the xrow 2 energy spectrum. The approximate nature of our measurement leads us to neglect the spread of particles in the y-direction; it is in any case a rather small correction.

The range of 5.5 MeV α-particles in P10 can be calculated from NIST range data. [10] This was done in [14]; the resultant range of α-particles in P10 is 4.14 cm. It is evident from the distribution of energies on xrow 1 that there are some particles that deposit at about as much energy on the second row as they do on the first; it is then evident that at least some particles traverse the entire pad, or come close to doing so. However, the vast majority do not, which is in conflict with the data from NIST.

This phenomenon can be explained by the nature of the radioactive source. The source has a finite thickness and is coated with gold for safety. The majority of α-particles from the source then lose energy by traversing the source itself and the protective foil. This explains the discrepancy in range, and indicates that our observations are what we might expect, at least approximately.

4.2.3 Interpretation of Results

The results shown above indicate that the DD-1 apparatus is able to self-trigger in case of a overhead, passing α-particle. Extensive work was done to verify that these were alpha particles, including:

- Verifying that the signal disappeared in the absence of a high voltage on the anode.
- Verifying signal attenuation in the absence of a drift field.
- Examining the rate of triggers with the chamber at voltage in the absence of a source.
- Examining the effect of source collimation on event rates.

Based on these facts, we may reason with some confidence that the board was successful in detecting alpha particles and, furthermore, giving crude information on their position. An examination of the $\chi^2_{\nu-1}$ distribution for the fits in Figure 4-10, however, indicates that there exists a difficulty in reliably fitting the coordinates. In the case of xrow 1, the basic shape of the χ^2 distribution is as expected, even though the magnitude of χ^2 is quite high. This indicates that errorbars for the data may have been chosen too small. However, this is clearly not the case for xrow 2, and this possible systematic error should be cause for future investigation.

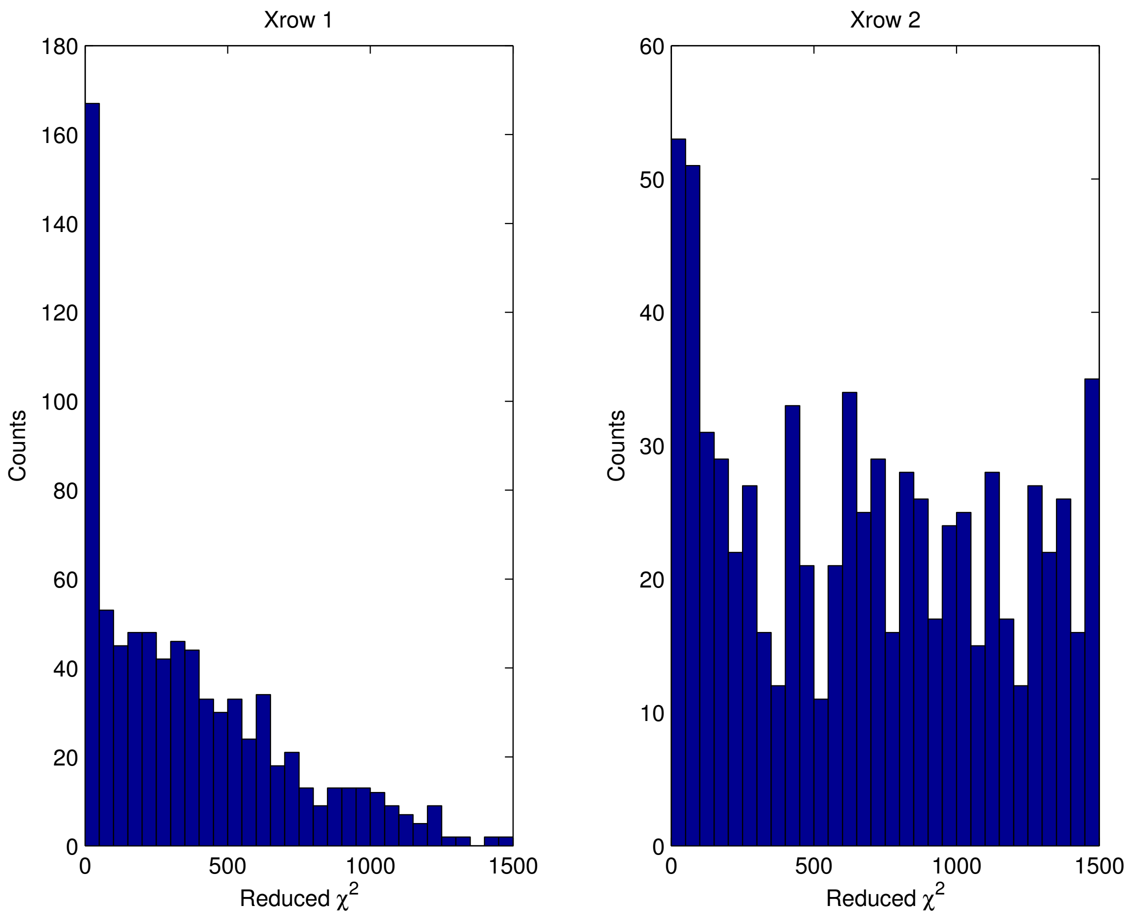


Figure 4-10: Reduced χ^2 values for 800 events, for each xrow.

Chapter 5

Conclusions

5.1 Summary of Results

In this thesis, we have demonstrated the functionality of a self triggering TPC readout that fits in one's hand. We have observed sensitivity to α-particle signals in the triggering regime, and have demonstrated a working readout, both in a testing environment and in chamber.

As we have mentioned before, this thesis does not represent the end of this project. Indeed, short- and long-term goals for the DD-1 project include the following:

- Improving the accuracy of the measured y-coordinate,
- Successfully demonstrating communication between modules,
- Successfully demonstrating multi-node track reconstruction, and
- Lowering the operating point of the chamber so as to not have to use α-particle sources.

This work is the first step in achieving these objectives.

Appendix A

DD-1 Readout Circuit Diagram

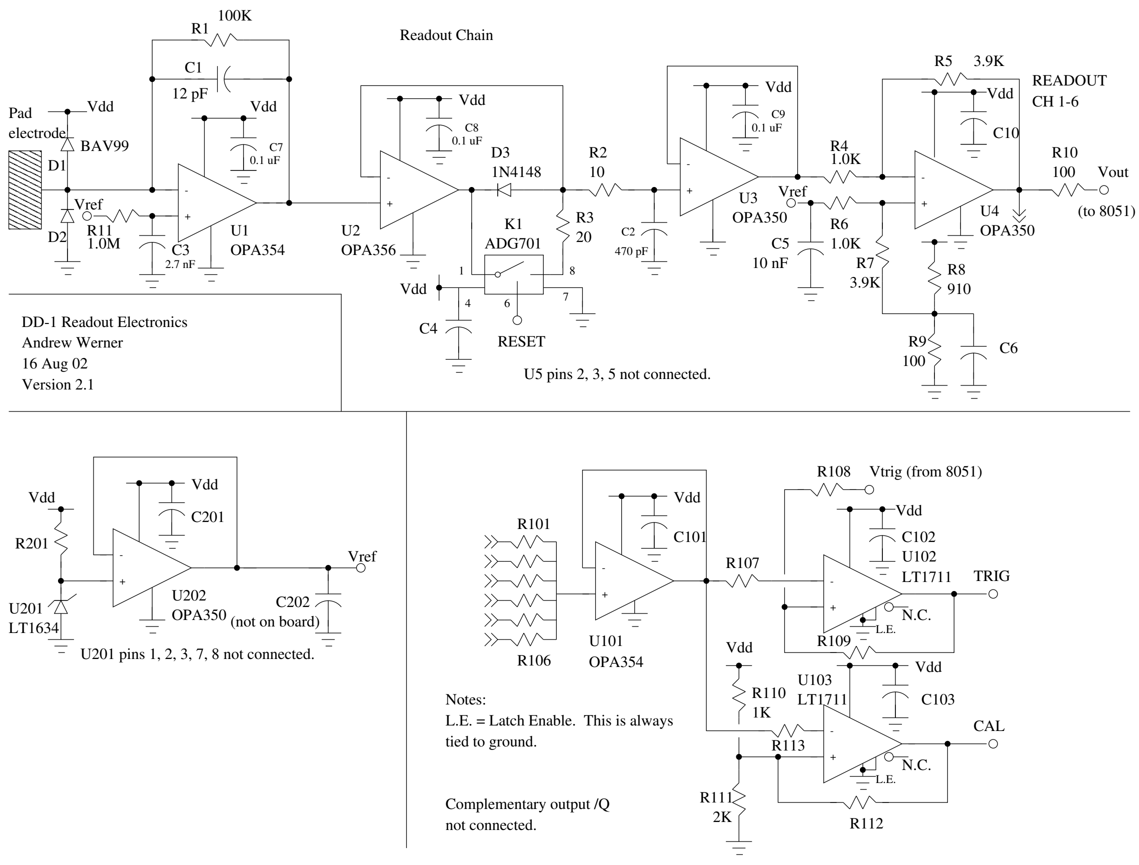
A.1 Circuit Design

In this Appendix, we describe the details of the design and operation of the DD-1 readout circuit. A circuit diagram is given in Figure A.1.

The first stage of each channel is an integrator that converts the current signal into a voltage signal that is proportional to the pad charge as a function of time. The feedback for the integrator consists of a resistor and capacitor in parallel, giving a characteristic time constant of approximately 10 μsec for the integrator. To accomodate the negative-going output signal from the integrator, the integrator stage is biased at 1.25 volts; this is derived from a shunt voltage reference (U201) on the board.

The second stage is a simple op amp-based negative peak detect circuit with reset, designed to minimize overshoot while still retaining high bandwidth. This, it should be noted, was done in the absence of a shaping stage for reasons of circuit size; a pulse shaping stage would have made the whole six channel apparatus prohibitively large. A version of this circuit was simulated in SPICE to give an idea of the component values required to give the circuit reasonable linearity with respect to peak input charge.

The final stage of the circuit is an gain of 4 output buffer which inverts the signal from the peak detect stage. The final buffer stage also changes the biasing on the


Readout Chain
R1 100K
C1 12 pF
Pad electrode
Vdd
BAV99
D1
D2
Vref
R11 1.0M
C3 2.7 nF
C7 0.1 uF
U1 OPA354
U2 OPA356
C8 0.1 uF
D3 1N4148
K1 ADG701
R3 20
RESET
C4
U5 pins 2, 3, 5 not connected.
R2 10
C2 470 pF
U3 OPA350
C9 0.1 uF
R4 1.0K
R6 1.0K
C5 10 nF
R7 3.9K
R5 3.9K
C10
READOUT CH 1-6
U4 OPA350
R8 910
R9 100
C6
R10 100
Vout
(to 8051)
DD-1 Readout Electronics
Andrew Werner
16 Aug 02
Version 2.1
R201
U201 LT1634
C201
U202 OPA350
C202 (not on board)
U201 pins 1, 2, 3, 7, 8 not connected.
R101
R106
U101 OPA354
C101
R107
R108
Vtrig (from 8051)
C102
U102 LT1711
TRIG
N.C.
L.E.
R109
R110 1K
U103 LT1711
C103
R113
CAL
R111 2K
R112
Notes:
L.E. = Latch Enable. This is always tied to ground.
Complementary output /Q not connected.

signal, from 1.25 volts to approximately 300 mV. Each channel output drives two different loads: an ADC input channel on the 8051, and an input to the summing comparator that begins the trigger stage.

Generation of the trigger signal is accomplished by averaging the output of each pad readout channel and then comparing the result to one of two voltages. This is illustrated in Figure A.1 as consisting of U101, U102, and U103. U101 sums all pad signals, and U102 and U103 are both high-speed comparators that undergo a high-to-low transition on overdrive. The trigger voltages are set differently for each comparator. U102 is set by the 8051 DAC, and is responsible for triggering on physical events. The voltage for U103 is set in hardware, and is set to be higher than would be reasonably expected for any signal due to an incident particle. An example use for this interrupt would be to signal all nodes for synchronization or calibration by simultaneously pulsing the sense wires with an overvoltage.

The operational amplifiers chosen for this application varied from stage to stage. The charge sensitive and peak detect stages were Texas Instruments type OPA356 and OPA354, respectively. This was primarily for the reason that the units ran off of a single ended supply, featured low bias currents, and had high bandwidth. The OPA350 was used for the output buffer stage, as it has lower bandwidth but higher DC accuracy after settling (which occurs on timescales of the order 100 ns).

The prototypes were assembled with multilayer surface mount technology by an outside contractor. PC Board design was by Sherilyn Harrison of Hanright Engineering Services. Assembly was by Argo Transdata Corporation.[1]

A.2 Performance

In order to verify proper board operation, a series of DC and AC tests were performed. DC testing verified board operation within manufacturers' parameters for all parts. Early AC tests verified the presence of a signal, and confirmed that the circuit does properly hold a peak.

[1]One Heritage Park, Clinton, CT 06413

The signal off the board exhibits a small amount of peaking due to the capacitive loading of the peak detect circuit.

For the comparator circuit using U102 (/TRIG), oscillatory behavior was obvserved for low overdrive levels. This is due to capacitive ringing in the VTRIG signal, from the 8051 board, that is triggered by the high-to-low transition on initial overdrive. This is most likely due to a circuit parasitic, specifically in regards to the VTRIG and /TRIG traces on the physical board. Device operation is not affected by this ringing, first because the /TRIG interrupt is edge triggered, and second because due to software masking of 8051 EX0, the software is insensitive to changes of the level of the /TRIG signals until the board is reset.

The AC testing was accomplished by observing the response of the circuit to current pulses at each of the pads. A pulser was connected via a capacitor to the pad input, and the pulses and capacitor were scaled so as to cover the dynamic fange of 300 fC - 6 pC. The steady state magnitude of the output (minus the pedestal value) was read off an oscilloscope. See Figure A-1.

The droop rate of the peak-hold stage was measured to be approximately 5 mV per 100 μsec.

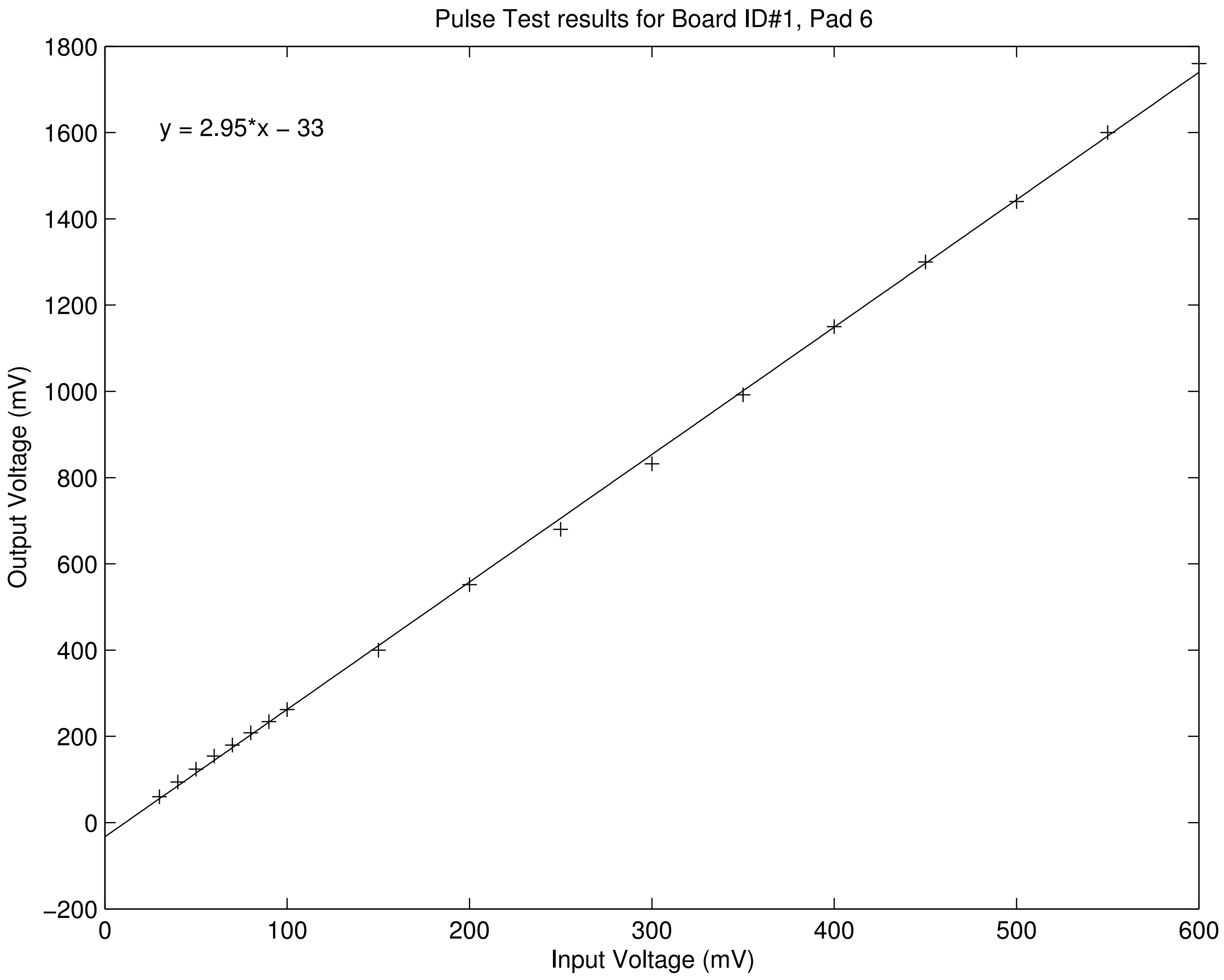


Figure A-1: Plot of steady-state output voltage versus input voltage for the DD-1 readout board.

Appendix B

Data Acquisition Procedure

We describe here the method by which data sets were acquired for this thesis.

B.1 Chamber Setup

1. Install the components of the drift chamber with the DD-1 readout: the DD-1 Readout board with Pushpin, the anode wires, the Frisch grid, and the drift cage, making all necessary electrical connections for correct chamber operation. Place a collimated ^{241}Am source in the volume above the readout pads in such a way that the projected particle tracks will traverse the six pads in the $\hat{x}$-direction.

2. Seal the chamber, using vacuum grease if necessary, and begin flowing P10 gas at a rate of 5-10 mL/sec. Continue the process of flowing the gas for at least fifteen minutes before proceeding.

3. Attach the anode and cathode high voltage supply wires to the appropriate chamber feedthroughs.

4. Open the CYGNAL IDE, and open the `chtest.wsp` project. Turn on the 3V DC supply to the Pushpin chamber, and begin running the `chtest` program via the JTAG interface.

5. The `chtest` program produces an animated visual display on the five board LEDs D401-D405, which is to be used as an indicator that the 8051 microcontroller is functioning correctly. With this animation running, bring the chamber voltages up to nominal levels, e.g. -600 V for the cathode and $+2.2$ kV for the anode.

B.2 Data Acquisition

1. In the CYGNAL IDE, open the `digitc2.wsp` project.

2. Connect to the board via JTAG, and download the program to the board. Add the variable `pedestals` to the watch as an array of `int` with index 5.

3. Begin running the program. At this point, D401 should be flashing at a rate of ~ 10 Hz, depending on the voltage level and the level of collimation. Each D401 flash corresponds to a trigger on the board.

4. After approximately 20 events have been collected (this should only take a few seconds), stop the running. Note pedestal values on the watch window.

5. From the Tools menu, choose the option "Upload memory to file." Choose a dummy text file for upload. Do a *decimal* dump of all of XDATA memory to the file.

6. Load the text file into MATLAB, and store the data as a vector:

```
> load data.txt
> v = data(:,1);
```

7. Use the MATLAB script `xerec` to create a vector of times and an $n \times 6$ matrix of pad values¿

```
> [times,pads] = xerec(v);
```

8. Repeat and concantenate data as necessary.

Appendix C

Source Code

C.1 C Programs

The following header files and programs were used in conjunction with the Cygnal IDE and Keil C Compiler for the 8051 microcontroller platform. [12, 6] Program function is given in the comments at the beginning of every program listing.

C.1.1 PushpinHardware

```
//////////////////////////////////////////////////////////////////
// PushpinHardwareV3.h revision A.1
//
// Josh Lifton
// MIT Media Lab
// Copyright 2001 all rights reserved.
//
// Some modifications due to particularities of interface between
// DD-1 readout hardware and Pushpins are included in this file.
// Modifications by:
//
// Andrew Werner
// MIT Laboratory for Nuclear Science
//
// Last modified 14 Aug 03 by awerner.
//
// Summary: (Lifton)
// This file specifies the hardware configuration for version 3 of
// the Pushpin architecture.
// The underlying processor is the Cygnal C8051F016.
//
/////////////////////////////////////////////////////////////////

// Port I/O assignments

// Pads defined as analog input channels

#define PAD_1 0x04
#define PAD_2 0x03
#define PAD_3 0x02
#define PAD_4 0x05
#define PAD_5 0x06
#define PAD_6 0x01

// System type definitions

typedef unsigned char OSErr; // Operating system error
typedef unsigned char Byte;  // A frequently used data type

#define noErr 0x00       // No error
#define qVFErr 0x01      // Queue overflow error
#define sBusyErr 0x02    // Serial busy error
```

```
// Port 0.

sbit txPin      =  P0^0; // Transmit pin for UART.
sbit rxPin      =  P0^1; // Receive pin for UART.
sbit TRIG       =  P0^2; // External event trigger input
sbit CAL        =  P0^3; // Calibration trigger input
sbit reset      =  P0^4; // Resets all peak-holds when high
sbit LEDY       =  P0^5; // LED, reference D403 (on when high)
sbit LEDGL      =  P0^6; // LED, reference D402 (on when high)
sbit LEDGH      =  P0^7; // LED, reference D404 (on when high)

// Port 1.

sbit LEDRH      =  P1^0; // LED, reference D405 (on when high)
sbit LEDRL      =  P1^1; // LED, reference D401 (on when high)
sbit comm0Pin   =  P1^2; // I/O pin on comm. connector
sbit statusLED  =  P1^3; // Red LED on processor board.
sbit comm1Pin   =  P1^4; // I/O pin on comm. connector
sbit comm2Pin   =  P1^5; // I/O pin on comm. connector
sbit comm3Pin   =  P1^6; // I/O pin on comm. connector
sbit comm4Pin   =  P1^7; // I/O pin on comm. connector.
```

C.1.2 PushpinUtil

```
/////////////////////////////////////////////////////////////////
// PushpinUtil.c
//
// Josh Lifton
// MIT Media Lab
// Copyright 2001 all rights reserved.
```

```
//
// Some modifications due to particularities of interface between
// DD1 readout hardware and Pushpins are included in this file.
// Modifications by:
//
// Andrew Werner
// MIT Laboratory for Nuclear Science
//
// Last modified 20 Jun 2003 by awerner.
//
// Summary:
// This file provides basic utility functions.
/////////////////////////////////////////////////////////////////

// Disable watchdog timer.
void configureWatchdogTimer() {
   WDTCN = 0xde;   // This register must be set first.
   WDTCN = 0xad;   // This register must be set second.
}

// Configure and enable the external 22MHz crystal.
void enableExternalClock() {
   CKCON |= 0x10;   // Don't divide by 12 for base clock
   OSCXCN |= 0x67;  // Start external oscillator > 6.74MHz.
   delay(0xffff);   // Wait more than a millisecond.
   while (!(OSCXCN & 0x80)) {} // Wait until ext. osc. is stable.
   OSCICN |= 0x80;   // Check for missing clock.
   OSCICN |= 0x08;   // Use external crystal.
}
```

```
// Configure crossbar and port I/O for general use.
void configurePorts() {
   PRT0CF = 0xFF;   // Set all of port 0 to push-pull outputs.
   PRT1CF = 0xFF;   // Set all of port 1 to push-pull outputs.
   PRT2CF = 0xFF;   // Set all of port 2 to push-pull outputs.
   XBR0 |= 0x04;   // UART TX and RX on P0.0 and P0.1.
   XBR1 |= 0x14;   // Setup /INT0 and /INT1
   XBR2 |= 0x40;   // Enable crossbar.
   XBR2 |= 0x80;   // Disable weak pull-ups.
}

// Configure and enable analog to digital converter.
// Select channel 0 as default.
void enableADC() {
   REF0CN |= 0x03;   // Enable VREF and ADC/DAC bias.
   AMX0CF = 0x00;   // Single-ended inputs (not differential).
   AMX0SL = 0x00;   // Select Analog Input 0 (AIN0).
   ADC0CF = 0x00;   // Gain 1.
   ADLJST = 0;   // Right justification.
   ADCTM = 1;    // Tracking is software controlled
                 // necessary for software control through ADBUSY
   EIE2 |= 0x02;   // Enable adc interrupt service.
   ADCEN = 1;   // Enable ADC.
}

// Initialize global variables.

void initializeVariables() {
statusLED = 0;
}
```

```
// Enable global interrupts, but first clear all interrupt flags.

void enableInterrupts() {
   TI = 0;   // Clear UART transmit interrupt flag.
   RI = 0;   // Clear UART receive interrupt flag.
   PRT1IF &= 0x0F;  // Clear IE4, IE5, IE6, and IE7 flags.
   CPT1CN &= 0x08;  // Clear comparator 1 interrupt flag.
   EA = 1;    // Enable all interrupts.
}

// Disable global interrupts.

void disableInterrupts() {
EA = 0;
}

// Enable P1.[4-7] as inputs and enable their external interrupts.

void enableCommChannels() {

   PRT1CF &= 0x0F;   // Set pins P1.[4-7] as open-drain.

   comm1Pin = 1;   // Set pin P1.4 as an input.
   comm2Pin = 1;   // Set pin P1.5 as an input.
   comm3Pin = 1;   // Set pin P1.6 as an input.
   comm4Pin = 1;   // Set pin P1.7 as an input.

   PRT1IF &= 0x0F; // Clear IE[4-7] interrupt flags.
   EIE2 |= 0x3C;// Enable interrupts IE[4-7].

}

// A simple delay.

void delay(unsigned int counter) {
while (counter--) {}
}

// Flashes the status LED a specified number of times
// at a given interval.

void flash(unsigned int flashes, unsigned int interval) {
   while (flashes--) {
      statusLED = 1;
      delay(interval);
      statusLED = 0;
      delay(interval);
   }
}

// Configure and enable digital to analog converter 0.

void enableDAC0() {
   REF0CN |= 0x03;   // Enable VREF and ADC/DAC bias.
   DAC0CN |= 0x07;   // Set DAC0 data format
                     // (lower nybble of DAC0L unused).
   DAC0CN |= 0x80;   // Enable DAC0.
   setDAC0Value(2048);   // Set the default value of DAC0.
```

```
}

// Sets the 12-bit DAC0 value given a 16-bit value and returns
// the new DAC0 value.  Note that DAC0L must be set before DAC0H
// for the full 12-bit value to latch.  Assumes the data format
// of DAC0 is such that the least significant nybble of DAC0L
// is unused.

void setDAC0Value(unsigned int DACvalue) {
   // DACvalue must fall within the DAC's 12-bit resolution.
   if (!(DACvalue > 0x0FFF)) {
      // Clear most significant nybble of DAC0L.
      DAC0L &= ~(0xF0);
      // Note that the MSN of DAC0L is actually
      // the LSN of the DAC setting.

      // Set LSN of DAC setting.
      DAC0L |= ((unsigned char) DACvalue) << 4;
      // Set MSB of DAC setting.
      DAC0H = (unsigned char) (DACvalue >> 4);
   }
}

// Returns the current 12-bit value of DAC0.  Assumes the data
// format of DAC0 is such that the least significant nybble of
// DAC0L is unused.

unsigned int getDAC0Value() {
   unsigned int DACvalue = 0;
   // Set the three MSNs of the return value.
   DACvalue = (unsigned int) DAC0H;
   DACvalue <<= 4;
   // Set the remaining LSN of the return value.
   DACvalue |= (unsigned int) (DAC0L >> 4);
   return DACvalue;  // Return the value of the 12-bit DAC.
}

// Configure and enable digital to analog converter 1.

void enableDAC1() {
   REF0CN |= 0x03;   // Enable VREF and ADC/DAC bias.
   // Set DAC1 data format (lower nybble of DAC1L unused).
   DAC1CN |= 0x07;
   DAC1CN |= 0x80;   // Enable DAC1.
   setDAC1Value(0x2900);   // Set the default value of DAC1.
}

// Sets the 12-bit DAC1 value given a 16-bit value and returns
// the new DAC1 value.  Note that DAC1L must be set before DAC1H
// for the full 12-bit value to latch.  Assumes the data format
// of DAC1 is such that the least significant nybble of DAC1L
// is unused.

void setDAC1Value(unsigned int DACvalue) {
   // DACvalue must fall within the DAC's 12-bit resolution.
   if (!(DACvalue > 0x0FFF)) {
      // Clear most significant nybble of DAC0L.
      DAC1L &= ~(0xF0);
      // Note that the MSN of DAC1L is
      // actually the LSN of the DAC setting.
      // Set LSN of DAC setting.
      DAC1L |= ((unsigned char) DACvalue) << 4;
      // Set MSB of DAC setting.
      DAC1H = (unsigned char) (DACvalue >> 4);
   }
}

// Returns the current 12-bit value of DAC1.  Assumes the data
// format of DAC1 is such that the least significant nybble of
// DAC1L is unused.

unsigned int getDAC1Value() {
   unsigned int DACvalue = 0;
   // Set the three MSNs of the return value.
   DACvalue = (unsigned int) DAC1H;
   DACvalue <<= 4;
   // Set the remaining LSN of the return value.
   DACvalue |= (unsigned int) (DAC1L >> 4);
   return DACvalue;   // Return the value of the 12-bit DAC.
}

// Configure and enable comparator 0.

void enableComparator0(void) {
   REF0CN |= 0x03;   // Enable VREF and ADC/DAC bias.
   CPT0CN |= 0x80;   // Enable comparator 0.
}

// Configure and enable comparator 1.
```

```
void enableComparator1(void) {
   enableDAC1();    // DAC1 is the inverting input to comparator 1.
   CPT1CN |= 0x80;  // Enable comparator 1.
   delay(1000);     // Wait for comparator to stablize.
   CPT1CN &= 0x08;  // Clear comparator 1 interrupt flag.
   EIE1 |= 0x80;    // Enable comparator 1 interrupt.
}
```

C.1.3 digitc

```
//////////////////////////////////////////////////////////////////
//
// digitc.c
//
// Extended version of histogram keeping file.  Logs events with
// limited time stamping, and computes pedestal values as is
// appropriate.
//
// Andrew Werner
// awerner@mit.edu
// MIT Laboratory for Nuclear Science
//
// Copyright (c) 2003 Andrew Werner.  All rights reserved.
//
//////////////////////////////////////////////////////////////////
#pragma SMALL INCDIR(C:\Code\PPLIB)

#include <c8051F000.h>
#include <PushpinHardwareV3.h>
#include <PushpinFunctions.h>
#include <PushpinUtil.c>
```

```
#define NUMBER_OF_EVENTS 20
#define THRESHOLD 620

// Definition of type Event:

typedef struct {
   unsigned long eventTicks;  // High order timekeeping bits
   unsigned int eventClocks;  // Lower order timekeeping bits
   unsigned char padValues[6];  // ADC counts on pads
} Event;

// Declaration of global variables

// Allows for indexing of ADC channels for each pad

code const unsigned char padArray[6] = ...
          ... = {PAD_1,PAD_2,PAD_3,PAD_4,PAD_5,PAD_6};

xdata Event resultBuffer [NUMBER_OF_EVENTS] _at_ 0x0000;

unsigned long systemTicks;  // Incremented on Timer3 overflow
sfr16 systemClocks = 0x94;  // Current Timer3 count
sfr16 timer3Reload = 0x92;  // Timer3 auto-reload value

unsigned int pedestals [6]; // Pedestal values, indexed by pad

// These bit variables are meant to be copied to
// the board LEDs after a pad is read.  They
// represent the high order nybble of the current
// 8 bit pad reading.

bdata unsigned char padReading;
sbit padReading_bit7 = padReading^7;
sbit padReading_bit6 = padReading^6;
sbit padReading_bit5 = padReading^5;
sbit padReading_bit4 = padReading^4;

bdata unsigned char flags;  // Reserved for future use

// Indicate whether we are acquiring pad data or a pedestal

enum status {IDLE, ACQUIRING_DATA, CALIBRATING};

unsigned char acquisitionStatus = IDLE;

int bufferPos;     // Our location in the event buffer
int currentPad;    // Current pad being read
int currentCalPad; // Current pad pedestal being read

// Function prototypes

void int0ISR (void);
void adc0ISR (void);
void tmr3ISR (void);
void initADC (void);
void initVariables (void);
void initInterrupts (void);
void initTimer3 (void);
void dispResult (void);
```

```
// Program begins here

void main() {
   configureWatchdogTimer();
   enableExternalClock();
   configurePorts();
   enableDAC0();
   setDAC0Value(THRESHOLD);
   initADC();
   initVariables();
   initTimer3();
   initInterrupts();
   while (1);   // We wait for an interrupt
}

void initADC (void) {
   AMX0CF = 0x00;   // Set all analog inputs independent
   AMX0SL = PAD_1;  // Select pad 1
   ADC0CF = 0x80;   // Set conversion speed, unity gain
   ADC0CN = 0x41;   // Software control, left justification
   ADCEN = 1;       // Enable the ADC
}

void initVariables (void) {
   bufferPos = 0;     // Start at beginning of event buffer
   currentCalPad = 0; // Look at the first pad being calibrated
   systemTicks = 0;  // Initialize the systemTicks variable
   P0 = 0x00;        // Zero both port outputs
   P1 = 0x00;
   reset = 0;    // Set peak detectors to peak-hold mode
}

// Sets up the system event timer

void initTimer3 (void) {
   TMR3CN |= 0x02;   // Set Timer3 to run at the system clock
   TMR3RLL = 0x00;
   TMR3RLH = 0x00;   // Set reload value at 0x0000

   TMR3CN |= 0x04;   // and start Timer3
}

// Interrupt initialization procedure specific for this application

void initInterrupts (void) {
   IE = 0x01;  // Turn on /INT0
   IT0 = 1;    // set as edge triggered
   IE0 = 0;    // and clear the pending flag.

   EIE1 = 0x00;
   EIE2 = 0x01; // Enable Timer3

   EX0 = 1;
   EA = 1;
}

// Interrupt service routine for a /TRIG interrupt
void int0ISR (void) interrupt 0 {
   LEDRL = 1;       // Turn on the trigger indicator LED
   currentPad = 0;  // Indicate we are reading pad 1
   AMX0SL = PAD_1;  // Select the first pad to readout
   EX0 = 0;         // Stop listening to /INT0
   EIE2 |= 0x02;    // Listen to ADC interrupts
   acquisitionStatus = ACQUIRING_DATA;  // Indicate status

   // Now timestamp the event
   resultBuffer[bufferPos].eventTicks = systemTicks;
   resultBuffer[bufferPos].eventClocks = systemClocks;

   ADBUSY = 1;  // And begin the ADC comnversion
}

void adc0ISR (void) interrupt 15 {
   ADCINT = 0;   // Clear ADC interrupt pending flag
   padReading = ADC0H;
   switch (acquisitionStatus) {
      case ACQUIRING_DATA:

      // Store conversion result in current event
      resultBuffer[bufferPos].padValues[currentPad] = padReading;

      // Display high order bits of result on node LEDs
      LEDGL = padReading_bit4;
      LEDY  = padReading_bit5;
      LEDGH = padReading_bit6;
      LEDRH = padReading_bit7;

      if (++currentPad == 6) {   // Are we done reading out?
         reset = 1;   // Yes we are: clean up
```

```
        delay(5);
        if (++bufferPos == NUMBER_OF_EVENTS)
           bufferPos = 0;
        acquisitionStatus = CALIBRATING;
        AMX0SL = padArray[currentCalPad];
        ADBUSY = 1;
     }
     else  // We're not done
     {
        AMX0SL = padArray[currentPad]; // Select next pad
        ADBUSY = 1;  // and begin conversion
     }
     break;

     case CALIBRATING:  // This is our pedestal tracking scheme
     pedestals[currentCalPad] = padReading;
     if (++currentCalPad == 6)
        currentCalPad = 0;
     reset = 0;
     acquisitionStatus = IDLE;
     EIE2 &= 0xFD;   // Disable ADC interrupts
     delay(10);
     LEDRL = 0; // Indicate we're done
     EX0 = 1;   // Enable ext. interrupts after things stabilize
     break;

     case IDLE:   // We should never see this
     break;
  }
}
```

```
void tmr3ISR (void) interrupt 14 {
   TMR3CN &= 0x7F;  // Clear Timer3 Interrupt flag (T3CON.7)
   systemTicks++;   // and indicate that a systemTick has passed
}
```

C.1.4 chtest

```
//////////////////////////////////////////////////////////////////
//
// chtest.c
//
// Creates a animated visual display on the DD-1 component side
// to verify correct 8051 operation.  Useful during the
// application of high voltage.
//
// Andrew Werner
// awerner@mit.edu
// MIT Laboratory for Nuclear Science
//
//////////////////////////////////////////////////////////////////

#pragma SMALL INCDIR(C:\Code\PPLIB)

#include <c8051F000.h>
#include <PushpinHardwareV3.h>
#include <PushpinFunctions.h>
#include <PushpinUtil.c>

void main() {
   configureWatchdogTimer();
   enableExternalClock(); // Perform minimal initialization
   configurePorts();

   while (1) {
      LEDRL = ~LEDRL; // Cycle all 5 DD-1 LEDs (D401-D405)
      delay(0xFFFF);
      LEDGL = ~LEDGL;
      delay(0xFFFF);
      LEDY = ~LEDY;
      delay(0xFFFF);
      LEDGH = ~LEDGH;
      delay(0xFFFF);
      LEDRH = ~LEDRH;
      delay(0xFFFF);
    }
}
```

C.2 MATLAB Scripts

C.2.1 padresp

```
%
% padresp.m
%
% function resp = padresp(y0,theta,wires)
%
% Takes in three vectors--one a vector of
% the values of y for which the pad
% response will be tested, a vector of the
% values of the track angle, and a vector
% of wire numbers for which to calculate
% the pad response.  Wire numbers are defined
% by the x-coordinate over which they are
% centered with respect to the center of the
% pad.  A wire with wire number w will be
% centered at x = x0 + wd, where d is the
% distance between sense wires.
%
% Andrew Werner
% 23 Jul 03
%

function resp = padresp(y0,theta,wires);

ly = length(y0);
lt = length(theta);
lw = length(wires);

x0 = 0;      % x-coordinate of wire zero
z0 = 4.8;    % z-coordinate of sense wire plane

d = 3;       % distance between sense wires

dx = 0.01;
dy = 0.01;   % determines precision of integration

l = 10;
w = 6;       % dimensions of pads
```

```
resp = 0*ones(ly,lt);  % initialize pad response vector

% Now we numerically integrate for all desired points

for k = 1:1:lw
  for j = 1:1:lt
    x1 = x0 + wires(k)*d*cos(theta(j));
    for i = 1:1:ly
      y1 = y0(i) + wires(k)*d*sin(theta(j));
      for a = (-l/2):dx:(l/2)
for b = (-w/2):dy:(w/2)
  resp(i,j) = resp(i,j) + ((dx*dy)/((a-x1)^2 + (b-y1)^2 + z0^2));
end
      end
    end
  end
end

% Normalize the array to the maximum

resp = resp/max(max(resp));
```

C.2.2 xerec

```
%
% xerec.m
% (eXtract Event RECord)
%
% function [t,p] = xerec(a);
%
% Takes in a vector of 8051 memory,
% and returns a matrix of pad events with a
% vector of times corresponding to the rows
% of the pad readings in the matrix.
%
% The event records as stored in memory are
% packed C structs:
%
% typedef struct {
%     unsigned long ticks;
%     unsigned int clocks;
%     unsigned char padValues[6]
% } Event;
%
% having a size of 12 bytes per events.
%
% Because of undocumented irregularities in the
% way in which data is stored by the 8051 C
% compiler across pages of XDATA, we look only
% at 20 events at a time.

function [t,p] = xerec(a); % t are times, p pad events

% Initialize our vector for analysis

temp = 0*ones(1,240);

% Look only at first 20 events

temp(1:240)=a(1:240);
```

```
% Form the matrix of pad events

p = 0*ones(20,6);
for i = 1:1:20
  for j = 1:1:6
    p(i,j) = temp((12*(i-1)) + 6 + j);
  end
end

% Form the column vector of times

t = 0*ones(20,1);
for i = 1:1:20
  t(i) = ((256^5)*temp(12*(i-1)+1)) + ((256^4)*temp(12*(i-1)+2)) + ...
      ((256^3)*temp(12*(i-1)+3)) + ((256^2)*temp(12*(i-1)+4)) + ...
      ((256^1)*temp(12*(i-1)+5)) + temp(12*(i-1)+6);
end
```

C.2.3 gaussfit

```
% gaussfit.m
%
% Finds the centroid of a matrix of xrows by
% performing a non-linear Gaussian fit to the
% each row of data.
%
% Andrew Werner
% 10 Oct 2003

% Vector a gives amplitudes for each fit,
% vector b gives the mean,
% and vector c gives the reduced chi^2.

function [a,b,chi2] = gaussfit(xrows);

size = length(xrows);

delta=7;

sig = 0.5*ones(3,1); % Set errorbars of 1/2 LSB
wgts = 1./sig.^2;
x=[-delta,0,delta]';

% Perform fit for each individual event

for i = 1:1:size
  y = xrows(i,:)';
  model=fittype('a*exp(-((x-b)/8)^2)');
  Value = 'NonlinearLeastSquares';
  opts=fitoptions('method',Value);
  opts.Weights = wgts;
  opts.StartPoint = [40,0];
  [fresult,gof,output] = fit(x,y,model,opts);
  a(i) = fresult.a;
  b(i) = fresult.b;
  chi2(i)=(gof.sse)/(gof.dfe);
end
```

C.2.4 sourcesim

```
% sourcesim.m
%
```

```
% function [m,b] = sourcesim;
%
% Simulates the angular distribution of $\alpha$
% particle tracks from test collimated source used
% in DD-1.  Tracks are represented in the form
% y = mx + b; we return vectors of m and b,
% respectively.
%
% Andrew Werner
% 26 Nov 2003

function [m,b] = sourcesim;

% Collimator specifications

holeRadius = 0.6477;
holeDepth = 6.858;
phiMax = atan(2*holeRadius/holeDepth);

% y-coordinate of opening in collimator

yOffset = 3.571;

numberOfTrials = 10000;
i = 1;  % Initialize our index for successful
        % Monte Carlo events

for ct = 1:numberOfTrials
  theta = 2*pi*rand;   % choose a random track direction
  phi = phiMax*rand;
  y0=2*holeRadius;
  z0=2*holeRadius;
  while (y0^2 + z0^2) >= holeRadius      % We try to choose
    y0 = 2*holeRadius*rand - holeRadius; % an origin for the track
    z0 = 2*holeRadius*rand - holeRadius;
  end
  y1=holeDepth*cos(theta)*tan(phi);
  z1=holeDepth*sin(theta)*tan(phi);
  if (sqrt(y1^2 + z1^2) < holeRadius) % Has the particle 'escaped'?
    b(i) = y0;                        % If so, we save the track
    m(i) = cos(theta)*tan(phi);
    i = i+1;
  end
end

b=b+yOffset;
```

Bibliography

[1] W. Blum and L. Rolandi. *Particle Detection with Drift Chambers.* Springer-Verlag, Berlin, 1994.

[2] W. Butera. *Programming a Paintable Computer.* PhD thesis, Massachusetts Institute of Technology, 2002.

[3] L3 Collaboration. Trigger and data acquisition system of L3. CERN, Geneva, 1984.

[4] P. Delpierre. The DELPHI TPC. *Nucl. Instrum. Meth.*, A225:566–575, 1984.

[5] H. Abelson et al. Amorphous computing. *Communications of the ACM*, 43(5), May 2001.

[6] Keil Software GmbH. `http://www.keil.com/`.

[7] W. R. Leo. *Techniques for Nuclear and Particle Physics Experiments*, chapter 2, 6. Springer-Verlag, Berlin, 2nd edition, 1994.

[8] J. Lifton. Pushpin computing: a platform for distributed sensor networks. Master's thesis, Massachusetts Institute of Technology, 2002.

[9] P. S. Marrocchesi et al. The spatial resolution of the ALEPH TPC. *Nucl. Instrum. Meth.*, A283:573–577, 1989.

[10] National Institute of Standards and Technology. ASTAR: Stopping-power and range tables for helium ions. Available on-line:
`http://physics.nist.gov/PhysRefData/Star/Text/ASTAR.html`.

[11] Cygnal Integrated Products. *C8051F0xx Family Datasheet.* Available on-line: `http://www.cygnal.com/datasheets/c8051f0xxx.pdf`.

[12] Cygnal Integrated Products. *Cygnal Integrated Development Environment (IDE).* Available on-line: `http://www.cygnal.com/support/developmenttools.htm`.

[13] F. Sauli. Principles of operation of multiwire proportional and drift chambers. Academic Training Programme of CERN, Geneva, 1977.

[14] J. Thompson. A readout system for a TPC detector. Senior Thesis, Massachusetts Institute of Technology, 2001.

[15] H. Wieman et al. STAR TPC at RHIC. *IEEE Trans. Nucl. Sci.*, 44:671–678, 1997.