Eindhoven University of Technology

MASTER

Pulse shape discrimination with a multi-parameter data-acquisition system

Simons, D.P.L.

Award date:
1994

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Master of Science Thesis

Pulse Shape Discrimination with a Multi-Parameter Data-Acquisition System

David Simons

October 1994

VDF/NK 94-45

Cyclotron Laboratory
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Prof. dr. M. J. A. de Voigt
Prof. dr. ir. K. Kopinga
Dr. S. S. Klein
Dr. L. J. van IJzendoorn
Ir. P. H. A. Mutsaers
Drs. A. J. H. Maas

Abstract

At the Nuclear Physics Techniques group an ion beam is used to perform material analysis. One of the analysis techniques is the Elastic Recoil Detection Analysis (ERDA) technique, which is applied for depth profiling light elements, like C,O and N, in a heavy matrix, like Si. It can be combined with Pulse Shape Discrimination (PSD) to separate the recoiled particles from the scattered He^{2+} projectiles that are also detected. For this purpose a detector with a thin depletion layer (*e.g.* 15 μm) is used. Recoils, which are stopped within the depletion layer produce fast output pulses because the charge they liberate is collected quickly. Scattered projectiles on the other hand are mostly stopped behind the depletion layer and the charge they liberate is collected more slowly by diffusion processes and the detector output pulse is slow. Discrimination between the recoils and the projectiles can be based on analysis of the detector output pulse shape.
A multi-parameter data-acquisition system has been developed during this master of science project that can be used to perform experiments with all experiment set-ups. To illustrate the usefulness and the power of this system ERDA-PSD experiments have been performed. Compared to conventional PSD, multi-parameter PSD gives a better separation between the contributions of the recoils and the scattered projectiles. Moreover, discrimation can be performed more accurately. This improves sensitivity and allows quantitative analysis of the measured data.

Contents

Introduction

This report gives an overview of the author's Master of Science project that was performed at the Nuclear Physics Techniques group. This group develops material analysis techniques and uses them to analyze samples like biological tissues and plasma deposited layers. Currently used analysis techniques are the Particle Induced X-ray Emission (PIXE) technique for trace element analysis, the ion scattering techniques Rutherford Backscattering Spectroscopy (RBS) and Elastic Recoil Detection Analysis (ERDA) for mass analysis and depth profiling and the channeling technique for structure analysis. These analysis techniques and the experiment set-ups in which these techniques are exploited are described in chapter 1.

Chapter 1 also describes the available data-acquisition computer system that consists of a *front-end* system for experiment control and data collection and of a *rear-end* system that stores, processes and monitors the experiment data. Finally, the need to develop software for this data-acquisition system, that will allow performance of multi-parameter experiments with *all* experiment set-ups, is described.

In chapter 2 the development of the software for the PhyDAS front-end data-acquisition system is described and in chapter 3 an outline is given of the specification, design and implementation of the new software, called Columbus 2, for the DEC Alpha rear-end computer system.

An application of the developed multi-parameter data-acquisition software is given in chapter 4. A number of ERDA experiments with Pulse Shape Discrimination (PSD) were performed to show that the use of a multi-parameter data-acquisition system improves the capabilities of this analysis technique.

Finally, chapter 5 gives conclusions and recommendations for future work.

Chapter 1
Techniques, set-ups and data-acquisition

In this chapter an overview is given of the way material analysis is performed by the EUT Nuclear Physics Techniques group. Section 1.1 describes the techniques that are used and in section 1.2 the experiment set-ups are discussed. In section 1.3 the data-acquisition system is treated and section 1.4 concludes with the proposed changes to this system.

1.1 Material analysis

At the physics department of the Eindhoven University of Technology a Philips AVF cyclotron is operated that can accelerate protons up to an energy of 26 MeV and alpha particles up to an energy of 30 MeV. The accelerated beam is used by the Nuclear Physics Techniques group to perform material analysis on samples like biological tissues or plasma deposited layers.
The beam is transported from the cyclotron to the several experiment set-ups by a beam guidance system. An overview of the experiment area is shown in figure 1.1.

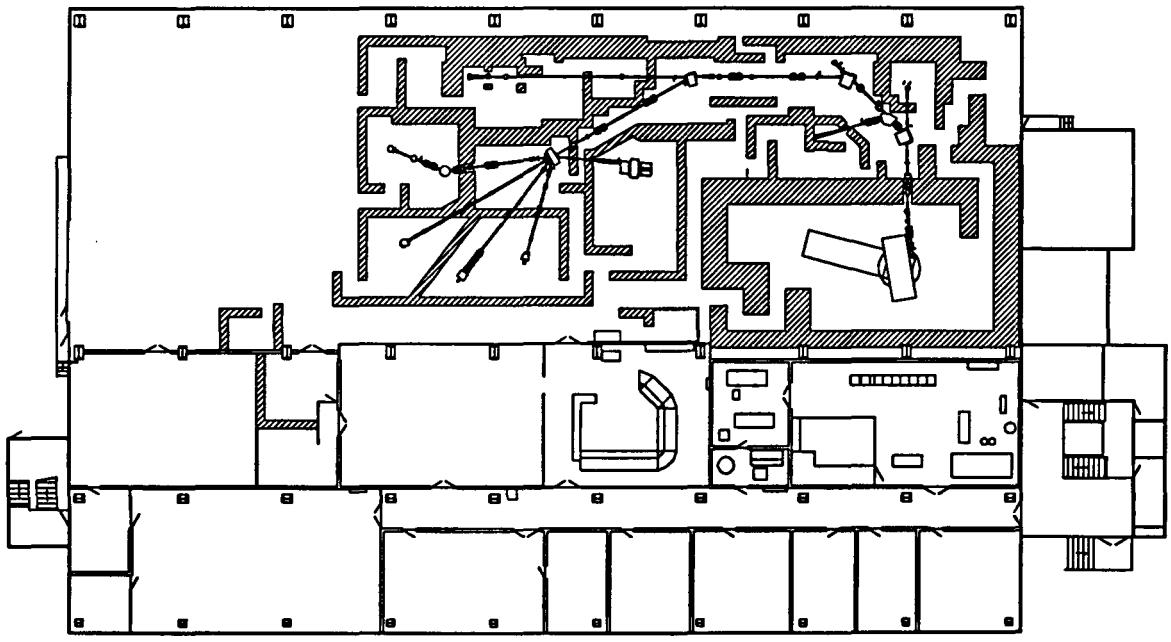

Figure 1.1: Layout of the ground floor of the Cyclotron building. The experiment area is located in the upper part of the picture. From the cyclotron at the right the beam guidance system leads the beam to the experiment set-ups

The analysis techniques that are currently used are the Particle Induced X-ray Emission (PIXE) technique for trace element analysis, the ion scattering techniques Rutherford Backscattering Spectroscopy (RBS) and Elastic Recoil Detection Analysis (ERDA) for mass analysis and depth profiling and the channeling technique for structure analysis. The experiment set-ups are a microbeam facility, an ion scattering

set-up and a channeling set-up. Both the techniques and the experiment set-ups will be discussed in the following sections.

1.1.1 The PIXE technique

The PIXE technique can be used for trace element analysis of samples. When a sample is bombarded with charged projectiles, usually protons, electrons are ejected from the target atoms. When an inner (K- or L-) shell electron is ejected the vacancy will be filled by an electron from a higher shell under the emission of X-ray radiation (fluorescence) or the creation of a so-called Auger electron. The energy of the X-ray photon is used to determine the chemical element of the excited target atom. The detection limit for an element is a few ppm (parts per million) and is limited by a/o low energy background radiation like bremsstrahlung.

1.1.2 Ion scattering techniques

Ion scattering techniques can be used for mass analysis and depth profiling of solid state samples. When bombarding a target an incident projectile can be scattered by a target atom and it can eject a target atom from the sample as a recoil. The ion scattering techniques are based on the detection of the scattered projectiles and recoils by measuring their energy, angle and/or speed. The Rutherford Backscattering Spectroscopy (RBS) technique is based on the detection of the backscattered projectiles and is most suitable for profiling heavy elements in a relatively light matrix. The Elastic Recoil Detection Analysis (ERDA) technique on the other hand, is based on the detection of the recoils and can be used for profiling light elements like H, N and O in a relatively heavy matrix.

The mass ratio μ of the masses of the target atom and the projectile can be determined in several ways. In conventional RBS and ERDA experiments the kinetic energies of the scattered projectile and the recoil are measured, respectively, while keeping the detection angles and the initial kinetic energy of the projectile fixed Then the measured kinetic energy depends on both the mass ratio μ and the depth d in the sample at which the collision occurred. The distribution of the projectiles initial kinetic energy over the scattered projectile and the recoil depends only on the mass ratio μ (and the type of collision: elastic of inelastic). When moving through the sample however, the particles lose an amount of kinetic energy through interaction with the other target atoms The amount of energy loss gives information about the depth at which the collision occurred and can be used for depth profiling. Notice that there is an ambiguity between the mass identification and the depth profiling, which is inherent to conventional RBS and ERDA.

A way to remove the mass/depth ambiguity and determine the mass ratio μ and the scattering depth d uniquely is Coincident Elastic Recoil Detection Analysis (CERDA). In this technique the kinematic relation between the scattering angle θ of the projectile and the scattering angle φ of the recoil is used to determine the mass ratio μ uniquely. During a CERDA experiment the angles φ and θ are fixed or θ is measured using a large position sensitive detector while keeping φ fixed. The measured kinetic energy will now provide the depth information.

Another way to determine the mass ratio μ and the scattering depth *d* uniquely is the expansion of CERDA with the Time-of-Flight technique (CERDA-TOF). As with the CERDA technique the mass ratio μ can be derived by measuring both the kinetic energy of the particles and their flight time over a fixed distance. When a mass selection has been performed, depth information can be obtained from the kinetic energy.
Another expansion to ERDA is the Pulse Shape Discrimination technique (ERDA-PSD) In this technique the recoils are separated from the scattered projectiles based on an analysis of the shape of the output pulse of the particle detector. This technique will be discussed further in chapter 4.

1.1.3 The Channeling technique

The channeling technique is used to perform structure analysis of crystalline solid state samples, like the study of crystal deformations and impurities. When the sample orientation relative to the beam axis is varied a channeling effect may occur when the ion beam is aligned with one of the crystal axes: the incoming ions are captured within and guided through the channels that are formed by the atoms along the crystal axis. This will cause a decrease in the yield of the backscattered projectiles. By studying the variations in the backscattered yield as a function of the sample orientation information can be obtained about the structure of the crystal and its deviations from an ideal crystal without deformations and impurities.

1.2 Experiment set-ups

The above analysis techniques are applied in three experiment set-ups: the microbeam facility, the ion scattering set-up and the channeling set-up. These three set-ups are discussed in the following sections.

1.2.1 The microbeam facility

In the microbeam facility a beam of accelerated protons is focused into a beam spot with a diameter of a few micrometers on the sample, using a pair of slits and a magnetic lens system consisting of four quadrupole magnets. A schematic overview of the experiment set-up is shown in figure 1.2.

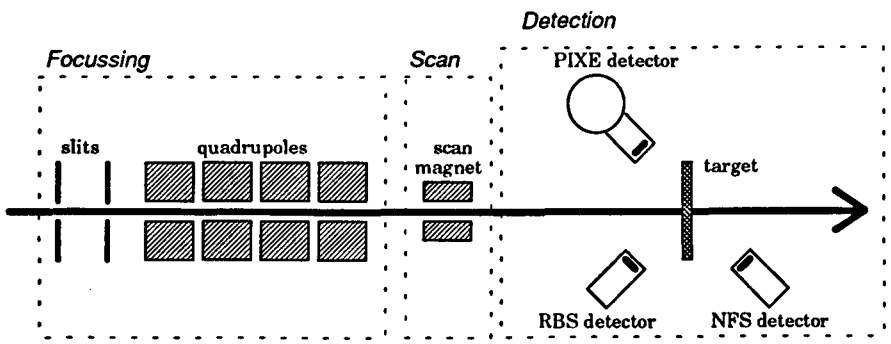

Figure 1.2: A schematic overview of the microbeam experiment set-up (Not drawn to scale. Actually the slits are at a distance of 6 m)

The two slits are located at a distance of 6 m from each other and determine the object size and the divergence of the incoming beam. After these slits the beam passes 4 quadrupole magnets, each focusing the beam in one direction and defocusing the beam in the other direction. By choosing out of a group of suitable combinations of quadrupole strengths, a net focusing effect is obtained and the beam size is decreased to a few micrometers in diameter.
Behind the focusing system a scan magnet is located that can deviate the beam and alter the position of the beam spot on the sample. The target wheel in which up to eight samples can be placed, can also be moved in the directions normal to the beam axis to change the position of the beam spot on the sample. The target holder is located in a vacuum chamber in which also three detectors are positioned. A X-ray detector that is used in combination with the PIXE technique and a RBS detector for detecting backscattered projectiles are located backwards and a NFS (Nuclear Forward Scattering) detector for particles scattered in a forward direction is located behind the target.

1.2.2 The ion scattering set-up

The ion scattering set-up is used for performing various types of ion scattering experiments, usually with alpha particles as projectiles. In figure 1.3 a schematic overview of the ion scattering experiment set-up is shown.

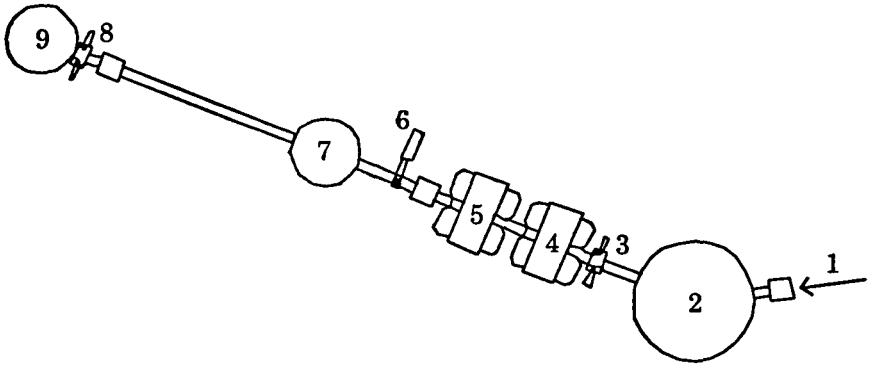

Figure 1.3: A schematic overview (top view) of the ion scattering experiment set-up. 1) incoming beam 2) scattering chamber 3,8) slit 4,5) quadrupole magnet 6) valve 7,9) detector chamber.

The main part of the ion scattering set-up is the scattering chamber (indicated by 2 in figure 1.3). The accelerated particle beam coming from the cyclotron via the beam guidance system enters the scattering chamber as indicated by 1 in figure 1.3. The beam will be aimed at a sample in the target holder. This target holder can be rotated so the angle of incidence of the beam on the sample can be varied.
Detectors and other objects like slits can be placed in the scattering chamber under all angles (both forward and backward) with an accuracy of a fraction of a degree and at different distances from the sample.
Furthermore a flight pipe is connected to the scattering chamber, at present fixed at an angle of 30 degrees with the beam axis, for performing CERDA-TOF measurements. A

detector can be placed at a distance of about 2 m (in the detector chamber as indicated by 7 in fig. 1.3) and 3.5 m (in the chamber as indicated by 9) from the sample. The flight pipe also contains two pairs of slits (indicated by 3 and 8), a valve (indicated by 6) and a pair of quadrupole magnets. These magnets can be used as a lens system for focusing recoils onto the detector located at the end of the TOF-pipe.
All kinds of detectors can be used in this set-up, e.g. particle detectors giving a signal that correlates to the energy of the detected particle or position sensitive detectors that give, in addition to the energy signal, one or more signals corresponding to the position where the particle hit the detector.

1.2.3 The channeling set-up

To apply the channeling technique a set-up was build consisting of an ultra high vacuum chamber containing a manipulator, also called goniometer, that is capable of rotating a (crystalline) sample in all three directions with steps of less than 1/100 degrees. The sample can also be moved in the directions normal to the beam axis.
A detector located at a backward angle is used to detect the backscattered projectiles. With this detector a/o the backscattered yield can be measured as a function of the sample orientation.
A second detector is used to detect the particles that are backscattered on a rotating vane that is located in the beam guidance system just before the vacuum chamber. The signal of this detector if used for beam current measurements to normalise the signals from the RBS detector.

1.3 Multi-parameter data-acquisition

In general, when performing experiments using the techniques and experiment set-ups as described above, a number of signals has to be measured, like energy signals from detectors, position signals from position sensitive detectors or from the scan magnet in the microbeam set-up of from the manipulator in the channeling set-up. These signals that have to be measured are called the experiment parameters. When during an experiment more than one parameter has to be measured it is called a multi-parameter experiment. Some of these parameters may be coincident, *i.e.* they are generated from the same collision event.
For collecting the data for all parameters, *i.e.* data-acquisition, a computer system is used to process all data coming from the experiment, the so-called data-flow. In sections 1.3.1 and 1.3.2 the different ways data can be collected in a multi-parameter experiment are discussed and in sections 1.3.3 to 1.3.6 the computer system and the way data is handled by it is discussed

1.3.1 List mode measurements

Measurements of the values for all parameters for every event may be collected in a (very long) list. This way of data-acquisition is called list mode and is the most powerful one because all single event data is stored independently without data reduction. A consequence is a large amount of data that has to be handled and stored. The advantage

of list mode data-acquisition is the possibility of extensive data processing afterwards, allowing the study of relations between parameters.

1.3.2 Histogram mode measurements

When we are not interested in direct relations between the parameters for every single event we can work in histogram mode. In this mode a histogram (spectrum) is collected for every parameter over a pre-set time interval or number of events. Now all single event relations are thrown away resulting in a reduction of the data flow in comparison to list mode.
It is up to the experimenter to decide whether to measurements have to be done in list mode (and keep the possibility of creating the histograms from the list mode data afterwards) or if the histogram mode will suffice.

1.3.3 PhyDAS

The so-called *front-end* part of the data-acquisition system is a Physics Data Acquisition System (PhyDAS) that has been developed at the Physics department of the EUT. PhyDAS can be divided into two parts: a PhyBUS part and a commercial VME bus part, connected to each other by a converter. A schematic overview of PhyDAS is given in figure 1.4.

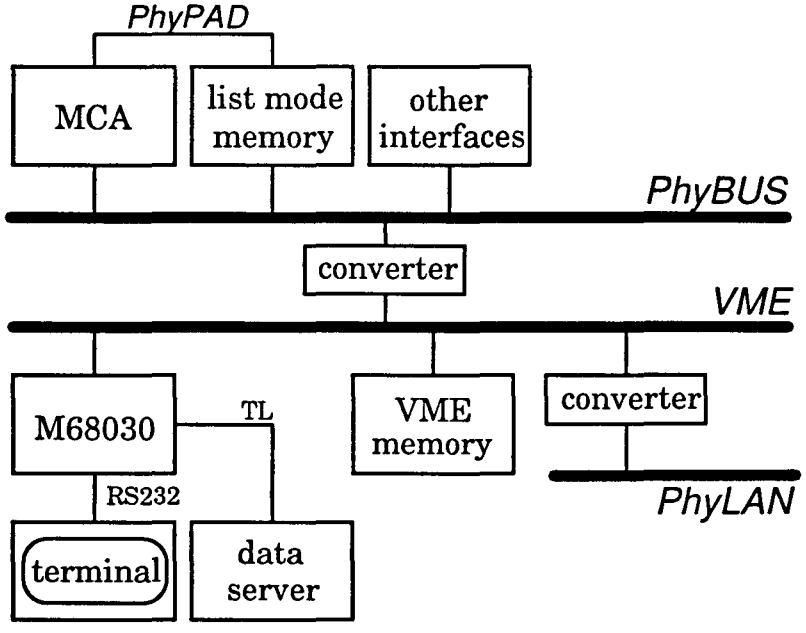

Figure 1.4: An overview of the PhyDAS data-acquisition system

The PhyBUS is the interface bus to which the interfaces for the control of experiment set-up, like current sources for magnets, stepping motors of target holders etc. are connected. Multi-Channel Analyser (MCA) interfaces and list mode memory boards that are used for the data-acquisition of analogue experiment parameters are also connected to this bus. The MCA board and the list mode memory board are discussed further in section 1.3.4.

Interfaces can be connected directly to each other by the Physics Parallel Asynchronous Data way (PhyPAD) for direct communication without burdening PhyBUS. This option is used for a/o the coupling between the MCA boards and the corresponding list mode memories.
In the VME bus a 25 MHz Motorola 68030 microprocessor (M68030) is located together with a 16 Mbyte VME memory board and a converter that connects PhyDAS to the Physics Local Area Network (PhyLAN) giving access to a file server for retrieving the operating system for the M68030 into the VME memory together with user software programs. PhyDAS can be controlled from a terminal that is connected to the M68030 using a serial RS232 line.
The currently used operating system is PEP030, which includes a line editor and a program interpreter for writing user software programs. Using so-called shell levels new commands, functions and variables can be added to the operating system.
Because the serial connection to the terminal is not fast enough for fast transfer of large amounts of data, this can be done through a transputer link (TL) that can be connected to other computers.

1.3.4 Multi-channel analysers and list mode memories

A schematic overview of a Multi-Channel Analyser interface is given in figure 1.5.

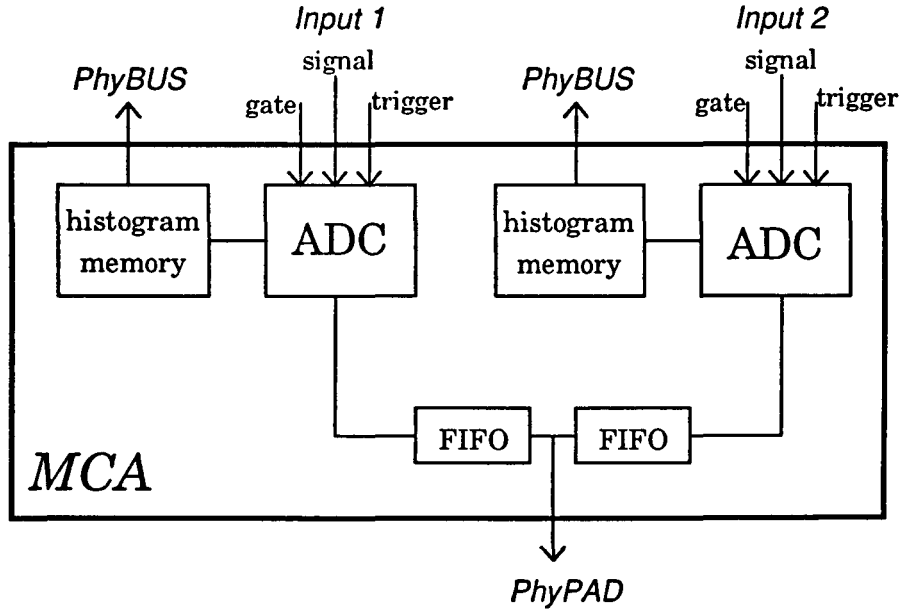

Figure 1.5: A schematic overview of a Multi-Channel Analyser board

A MCA interface board contains two Multi-Channel Analysers with each a 12-bit analogue-to-digital converter (ADC). When a pulse arrives at the trigger input of an ADC it will convert the analogue input signal (range 0 through 10 V) into a digital word (range 0 through 4095). A gate signal indicates whether the converted digital word should be regarded as valid. Both MCAs have their own (32 bit) histogram memory in

which they can register how may times each digital value occurred. This histogram memory can be accessed via PhyBUS.
When the PhyPAD output of the MCA board is enabled, the converted digital values are also sent through FIFO (First In First Out) buffers to the PhyPAD output. If only one MCA is active (single MCA mode) the words sent to PhyPAD will be 16-bit words (of which the lower 12 bits contain the ADC data and bit 15 is set when the word should be regarded invalid). When both MCAs are operating (dual MCA mode) their 16-bit words are combined into a 32-bit word before they are sent to PhyPAD. Notice that the ADC's should both be triggered in order to have a 16-bit value from each ADC that can be combined into a 32-bit word.
The PhyPAD output of the MCA board is connected to the PhyPAD input of a list mode memory board. This memory board will store the digital words coming from the MCA board in list mode. The memory will operate in 16-bit mode or in 32-bit mode, according to the mode in which the corresponding MCA board works.
The PhyBUS also contains memory boards that are not connected to the PhyPAD output of a MCA board but to the PhyPAD output of other interfaces that provides digital words *e.g.* the scan pattern generator in the microbeam facility that provides 16-bit position identification numbers via PhyPAD.

1.3.5 Data transmission and storage

When data has been collected by PhyDAS it has to be transferred to the *rear-end* computer system where it can be processed. The *rear-end* system is a Digital AXP workstation (Alpha) running the VMS the operating system and an X-Windows user-interface. Connected to it are a/o several 1 Gbyte hard disks.
Because the transputer link from PhyDAS cannot be connected directly to the Alpha, a data server is used to realise the coupling. A schematic view of this coupling is shown in figure 1.6.

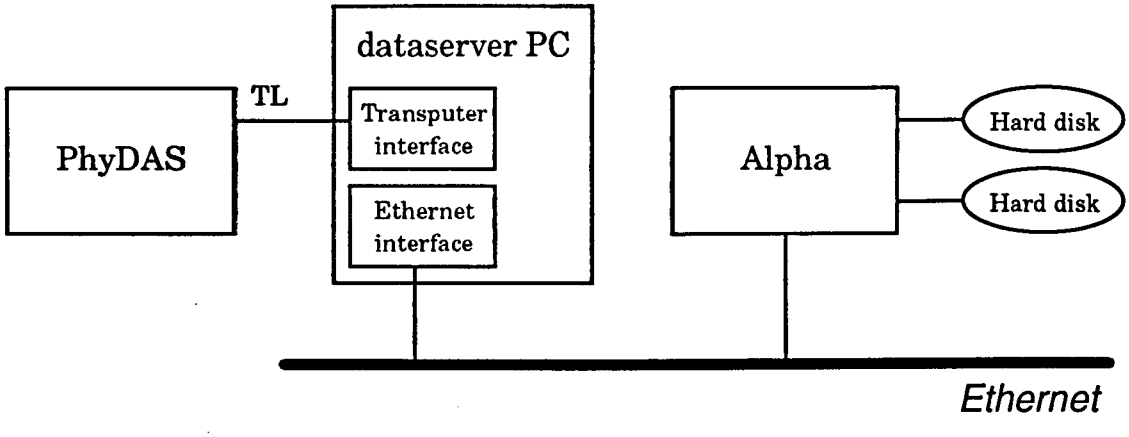

Figure 1.6: Schematic overview data transmission from PhyDAS to the Alpha.

The PhyDAS system is connected via the transputer link to a transputer interface that is located in a personal computer acting as the data server. A data server program running on this PC will allow PhyDAS to write data into files on a virtual hard disk.

This might be the local hard disk in the data server PC, but this disk cannot be accessed by the Alpha workstation. This problem is solved by adding one of the hard disks of the Alpha as a virtual hard disk to the data server PC, using Ethernet.

1.3.6 Columbus

When the list mode or histogram mode data is stored into files on the Alpha hard disk it is available for applications to process it. One of these applications is Columbus which has been developed for processing data coming from the microbeam experiment set-up. It has an X Windows based Graphic User Interface and an example of this interface is given in figure 1.7

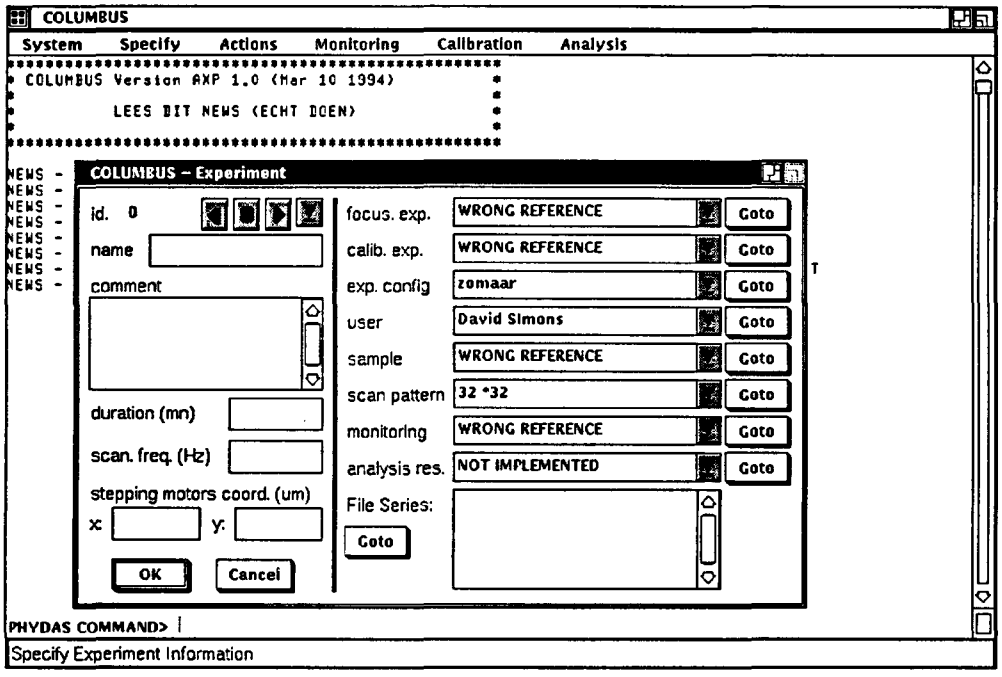

Figure 1.7: The X Windows GUI of Columbus 1. The main window is shown along with one of the window for editing the data base.

Columbus consists of four modules:

1. Data base module
2. Experiment control module
3. Monitoring module
4. Off-line analysis module

ad 1.: The data base is the central module that is used by the other modules. For every experiment that is performed information about the set-up, user, samples, ion beam properties can be stored together with the monitoring demand, *i.e.* what graphics (graphs or plots of measured data) should be displayed during an

experiment. Furthermore, results from off-line analysis, *i.e.* analysis afterwards, may also be stored in the data base.

ad 2.: Columbus should be able to control the experiment by giving commands to the PhyDAS data-acquisition system and its interfaces. Experiment control by Columbus has not yet been fully implemented because there is no direct interaction possible between Columbus and PhyDAS. Experiment control is now done by the user using the terminal connected to PhyDAS.

ad 3.: During a experiment Columbus provides the possibility to display graphics containing graphs or plots of the measured data. This is called experiment monitoring. When PhyDAS has written the collected data into files on the Alpha hard disk, Columbus can access these files, read the data from it and process it into the graphics. So Columbus will not monitor the data real-time, *i.e.* following the experiment at the same pace with a fixed delay, but on-line, *i.e.* as fast a possible. The delay between the collection of the data by PhyDAS and the display by Columbus depends on the frequency by which PhyDAS writes the data to the hard disk (the so-called sample time) and on the speed by which Columbus can process the data into graphics.

The transmission of the data using files on hard disk can also be seen as an advantage: Data can be re-monitored afterwards, as if the experiment was in progress, whereby different graphics can be created by changing the monitoring demand in the data base. This may seem like off-line analysis; the monitoring should, however, not be expected to give final analysis results. It is meant to give an overview of the experiment progress and of possible results.

ad 4.: The off-line analysis part of Columbus contains applications that can process the measured data together with information from the data base into final results. These analysis results may then also be added to the data base.

1.4 Changes in the data-acquisition system

Both Columbus and most of the PhyDAS data-acquisition software CEDAS (Cyclotron Eindhoven Data Acquisition Software) have been developed for the microbeam facility and its experiments. With CEDAS both list mode and histogram mode experiments can be performed, but only for measuring the fixed set of parameters coming from the microbeam set-up. General multi-parameter experiments are not possible and coincidences between parameters are disregarded. Furthermore, the several modules of Columbus are fully dedicated to the microbeam experiment set-up.

So if we also want to use the combination of PhyDAS and Columbus for multi-parameter experiments with the ion scattering experiment set-up and channeling experiments with the channeling experiment set-up changes have to be made to both the PhyDAS data-acquisition software and Columbus. This has been done during the first part of the Master of Science project. The new data-acquisition software for the PhyDAS system is discussed in chapter 2 and the new version of Columbus that was designed and implemented is discussed in chapter 3.

Chapter 2
The new PhyDAS data-acquisition software

In this chapter the data-acquisition software for PhyDAS is discussed. Section 2.1 gives an overview of the existing software and the proposed changes to it. In section 2.2 the new software is described, including the method for defining a multi-parameter experiment set-up within PhyDAS. Finally, in section 2.3, new standards for the data transfer from PhyDAS to the Alpha workstation are presented.

2.1 Overview of the existing PhyDAS software

During the last years software was developed for performing data-acquisition with PhyDAS. For the microbeam facility software was developed to control the experiment set-up and to perform focusing experiments, beam diameter measurements and list mode experiments [ATZ92]. Later, the option of histogram mode experiments was added [SIM93]. This complete package of software is called Cyclotron Eindhoven Data Acquisition Software (CEDAS)

Independently, software was developed for the data-acquisition [JON93] and monitoring [STR93] of simple multi-parameter ion scattering experiments. For channeling experiments software independently developed to control the experiment set-up, perform data-acquisition and monitoring [BEU94].

Except for microbeam list mode experiments that were monitored using Columbus, monitoring of the measured data during experiments was unsatisfactory. The other experiments were monitored on a low resolution monitor connected to PhyDAS.

So the question arose whether there were possibilities to combine parts of these software packages into one general purpose program that could be used to control all experiment set-ups and to perform full (coincident) multi-parameter ion scattering experiments, microbeam experiments and channeling experiments. Furthermore, if it would be possible to use Columbus to monitor and analyse the data of all experiments. The answer to these questions turned out to be positive and the following actions had to be undertaken to obtain a uniform software package:

1. Modify the CEDAS software to make it support multi-parameter experiments, also with coincident parameters, in both list mode and histogram mode, so it can be used for all experiment types.
2. Change Columbus so it can monitor the data from experiments done with the new CEDAS software.
3. Integrate the software to control the channeling experiment set-up within CEDAS and modify it such that Columbus can be used for to monitor the channeling experiments as well.

As a consequence of these changes the existing multi-parameter software will be superfluous and all data-acquisition will be integrated into one package: CEDAS.

Action points 1 and 2 have been carried out during this Master of Science project and the results are discussed in sections 2.2 and 2.3, respectively.

2.2 The new PhyDAS multi-parameter software

In section 2.2.1 the changes to CEDAS are discussed that are required to make it support multi-parameter experiments. In section 2.2.2 an overview of the new software itself is given.

2.2.1 Multi-parameter set-up

For every multi-parameter experiment, a number of parameters have to be recorded by the data-acquisition system. To allow flexible multi-parameter experiments, a standard is created to define an experiment set-up within PhyDAS, to keep track of all parameters. The number of parameter has to be known and relations between parameters, like coincidences, have to be registered. Therefore the parameters are sorted out into so-called modules. We have to make a difference between list mode experiments and histogram mode experiments, in the definition of a module:

- List mode module definition
  During a list mode experiment the value of every parameter at every event has to be registered in one or more lists. We sort the parameters that are *coincident*: all parameters that are **coincident** go into **one module**. So one module only contains parameters that are coincident with each other. Parameters in one module are not coincident with parameters from another module.
  As a consequence, for every parameter in a certain module, the list length is the same.

- Histogram mode module definition
  When performing histogram mode experiments a histogram is recorded for every parameter in the experiment. The parameters of which a spectrum has to be recorded **during the same time interval** are put into **one module**. Notice that this does **not** require the parameters in one module to be coincident, because single event relations are not of interest in histogram mode.

Another difference between list mode and histogram mode experiments is the way in which data is stored during data-acquisition. In list mode we have to store lists of parameter values and in histogram mode we have to store spectra.

- List mode data storage

  The list mode memories are used to store the list mode data for each parameter. One memory board can contain the list of a single parameter (16-bit mode) or of two coincident parameters (32-bit mode). When the list mode memory is connected to a MCA board both the MCA board and the list mode memory should operate in the same mode. As already indicated the two parameters that are acquired by one MCA/memory combination in 32-bit mode should be coincident and thus originate from one module.

- Histogram mode data storage

  The histogram of every parameter is collected in the internal histogram memory of the corresponding MCA. The histogram memories of both MCAs in one MCA board operate completely independent when the PhyPAD output of the MCAs to the list mode memories is disabled. During histogram mode PhyPAD is indeed disabled to prevent the list mode memories from overload, so now the two parameters do not have to be coincident. They should be in the same module, however, because both MCAs in one MCA board are started and stopped at the same time.

After this, an algorithm for defining the multi-parameter set-up within PhyDAS can be defined. The parameters from every module have to be divided over the memories; during list mode measurements the list mode memory boards are used for data storage and during histogram mode measurements the internal histogram mode memories of the MCA boards are used. Because one definition that can be used for both list mode and histogram mode is favourable, the following algorithm is chosen.
First, the number of modules of the experiment set-up has to be determined. This is done with the use of the definitions of the parameter modules as given above.
Then, for every module the number of list mode memory boards has to be defined and for every memory board it has to be defined if it is used for storing the list of one parameter or two coincident parameters. This will determine the mode in which each memory board operates (16 or 32 bit). Furthermore, for every list mode memory it has to be indicated whether it is connected to the PhyPAD output of a MCA board or to the PhyPAD output of another interface board that provides digital data. This board is expected to operate in the same mode as the memory board. So the MCA mode (single or dual) of each MCA board is determined by the memory mode of the corresponding list mode memory board.
During histogram mode measurements, PhyPAD remains disabled and the mode in which the MCA boards operate determines what histogram memories are used.
This algorithm has successfully been implemented into the CEDAS software.

2.2.2 The level structure

The PhyDAS software CEDAS consists of several parts, called levels, that have to be loaded on top of each other. Upper levels can make use of procedures and variables from lower levels. Lower levels contain routines for *e.g.* interface control. These 'low level' routines simplify the use of interfaces in upper levels. Figure 2.2 shows an overview of the software levels.

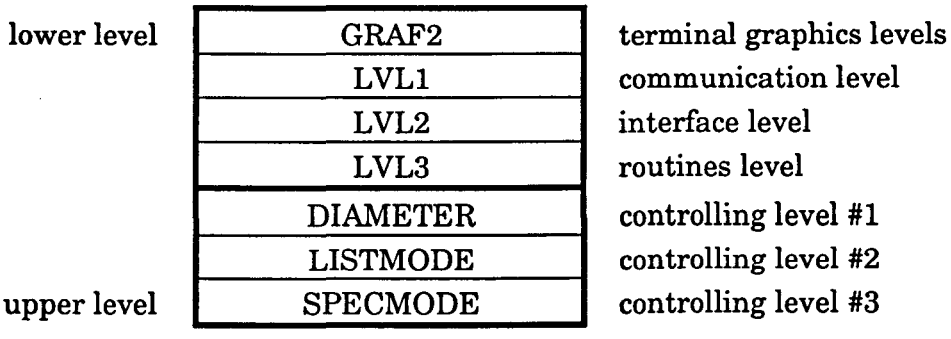

Figure 2.2: Overview of the PhyDAS software levels

The four lowest levels are general purpose levels.

- *GRAF2*
  This level contains routines for drawing graphics on the terminal.
- *LVL1* - The communication level
  The objective of this level is to simplify and speed up the communication between the PhyDAS system and the terminal through the serial RS232 connection. This is realised by decoding messages, errors and questions into numbered codes that are transmitted much faster.
- *LVL2* - The interface level
  In this level an abstraction is made from the hardware addresses and the interface control. Every interface is given an index, and via an interface table this index is related to the appropriate hardware addresses. Furthermore this level contains routines for starting/stopping interfaces, memory operations, MCA control etc.
- *LVL3* - The routines level
  In this level the control of the experiment set-up by interfaces is simplified. Currently, only part from the microbeam set-up, like the quadrupoles, the target wheel and the scan magnet, can be controlled
  Routines were added to define the multi-parameter set-up (the so-called module definitions). Furthermore, routines were added to start and stop all MCAs instantly by enabling and disabling the trigger inputs all at once with a relais interface. This option was necessary for coincidence measurements with more than one MCA board per module. (See also [JON93], p. 20).
  The routines for controlling the channeling set-up have to be added to this level.
- *DIAMETER* controlling level
  This level contains routines to (automatically) focus the beam and to perform beam diameter measurements with the microbeam set-up. In the future this level may be used for auto-focusing of the quadrupoles of the ion scattering set-up.
- *LISTMODE* controlling level
  Level for performing list mode measurements. It can now perform full multi-parameter experiments. The module definition from *LVL3* determines what parameters are to be measured.
- *SPECMODE* controlling level
  For performing histogram mode measurements. This level has also been modified to work with the new multi-parameter standard. The new file standard (section 2.3.2) has also been implemented.

In appendix A an overview can be found of all changes that have been made to the CEDAS software.

2.3 The new data transfer standard

A standard has been designed for transferring data from PhyDAS via the data server to the Alpha workstation. This standard is implemented in the CEDAS software and the software for performing channeling experiments. The new version of Columbus (chapter

3) will also accept this new standard and can thus be used to process the data from all experiments.
Two formats have been developed, one for list mode data and one for histogram mode data, both of which we are discussed in the following sections.

2.3.1 List mode file format

The list mode files should contain the data that is collected in the list mode memories for each parameter over some time interval and additional information like the measuring time, the percentage of dead time, the charge, the number of events in the list for each parameter and the memory mode.
To make the file transfer as fast as possible for each list mode memory a binary copy of its contents is put into a file together with a header containing the additional information. So one file contains the list of one or two parameters, depending on the memory mode of the corresponding list mode memory. Notice that the file header for each file corresponding to a memory in the same module has to be equal. A complete description of the list mode file format is given in appendix B.

2.3.2 Histogram mode file format

A histogram file contains the contents of one histogram memory, thus the spectrum of only one parameter. This is more flexible than putting all histograms into one file. It was decided to make the files ASCII-text-files to keep them compatible with earlier standards (*e.g.* AXIL).
In addition to the histogram data each file also contains additional information in a header about the measuring time, the current during the collection of the spectrum, a possible scan pattern etc. A complete description of the histogram mode format is given in appendix C.

Chapter 3
Specification, design and implementation of Columbus version 2

This chapter describes the development of the new version of Columbus. A list of microbeam requirements for this Columbus 2 is given in section 3.1. The channeling requirements are listed in section 3.2 and the ion scattering requirements in section 3.3. Based on these requirements a design was made for the new monitoring module (section 3.4), the new data base module (section 3.5) and the corresponding user-interface (section 3.6). In section 3.7, an overview of the implementation of this design is given. Finally, in section 3.8 a list of suggestions for future work is presented.

3.1 Microbeam requirements

Most of the requirements of the microbeam set-up for Columbus 2 are already realised in the first version of Columbus (Columbus 1) of which a full description can be found in reference [ZWI93]. Columbus 2 will thus have to be compatible with Columbus 1. Some additional requirements have to be added. All microbeam requirements are listed in the following sub-sections.

3.1.1 The data base

The objective of the data base is to replace the conventional logbook. All information that is of interest about the experiment, in addition to the acquired data, should be stored in the data base.

A microbeam experiment session roughly consists of three parts. First, a focusing experiment is done to minimize the beam spot size on the sample down to a few μm^2. Then a calibration experiment is done before the main experiments can be performed. For all of these experiments it should be possible to store:

- information about the experiment set-up (number, type, location and composition of detectors)
- information about the experimenter (name, status, phone)
- information about the samples (analysis demand, thickness, photos, client, position in target holder)
- information about the scan pattern (shape, size, frequency)
- monitoring demand (number of graphics and type of each graphic)
- analysis results of the off-line analysis
- information about the measured data (location, number and type of files from PhyDAS)
- general information (date, beam properties)

3.1.2 The monitoring module

The monitoring module can handle list mode data coming from up to three detector modules (PIXE, RBS, NFS). Every module contains two parameters: the energy signal

from the detector and a position identification number (PIN) from the scan pattern generator corresponding to a position of the beam on the sample. Columbus can sort this list mode data into two types of graphics:

1. Energy histogram
   The energy list mode data from a module is sorted out into a histogram where limits can be set on the corresponding PIN data (actually on the x- and y-co-ordinates that are derived from the PINs). So a position dependent energy spectrum is obtained. An example of such a histogram is given in figure 3.1.

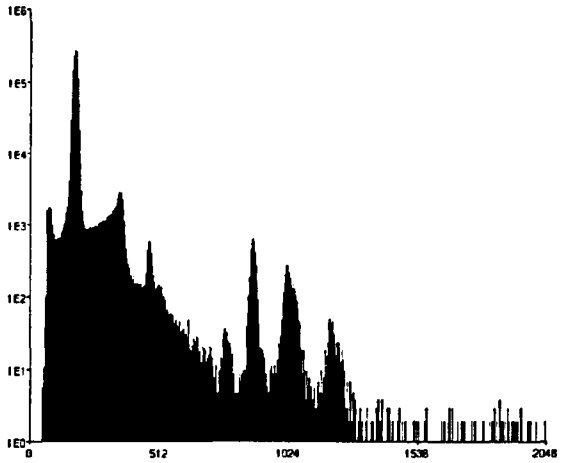

Figure 3.1: An PIXE energy histogram graphic created with Columbus 1. A W/Ti chip was bombarded with 3 MeV protons. The yield is plotted versus the energy channel number.

2. Distribution picture
   The position list mode data from a module is sorted out into a distribution picture where limits can be set on the corresponding energy values. So an energy dependent distribution picture is obtained. Figure 3.2 shows an example of a distribution graphic.

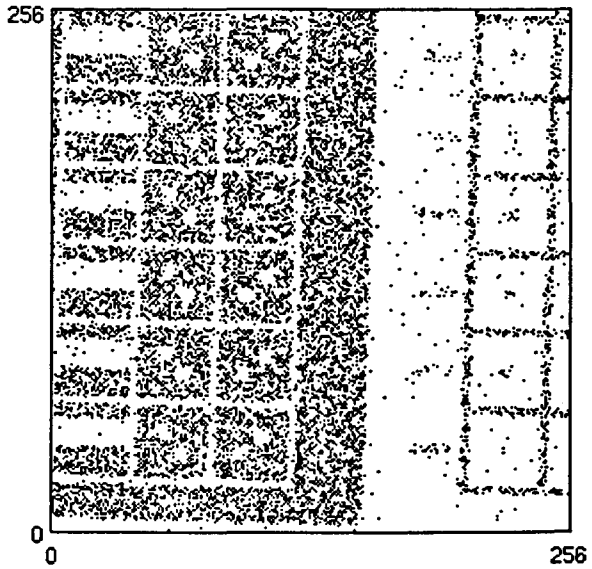

Figure 3.2: Distribution graphic created with Columbus 1. The x- and y-position on the sample are plotted along the horizontal and vertical axis, respectively. The intensity corresponds to the yield The contours of the W/Ti chip are clearly visible

3.2.3 The off-line analysis

In addition to the monitoring module that will provide on-line information about the current experiment, off-line analysis is used to process the measured data into final results like concentration distributions for trace elements. From within Columbus several off-line analysis programs can be started. These programs are provided data from the data base. Current off-line analysis programs are:

- SortLMData
  A program to sort list mode data into spectra. It has more options than the monitoring module, e.g. channel compression, and can be run in batch (background) mode. Unfortunately, this program is only capable of sorting out microbeam list mode data.
- AXIL
  Program for the analysis of PIXE energy spectra.
- Egg
  Off-line analysis program for calibration and analysis of RBS and NFS energy spectra.

3.1.4 New demands

Some new microbeam requirements have to be added to the list of existing demands:

- The possibility to monitor histogram mode experiments.
- The possibility to scale spectra to for example the beam charge.
- The option of graphics with time as a parameter, to monitor for example radiation damage measurements.
- Option to save graphics that are created by Columbus, so they can be re displayed later without having to sort the list mode once again.
- The option to handle list mode data of experiments with coincident parameters

3.2 Channeling requirements

3.2.1 Data base

The channeling demands for the data base are roughly the same as for the microbeam experiments. The data base should contain all experiment data like set-up, user and beam properties. There are however some differences:

- The channeling set-up does not have a scan magnet to move the beam across the sample. A scan-pattern might be used to move the sample in a pre-defined pattern with the manipulator.
- More data about the detectors in the set-up is required, because the location, angle and distance, are not fixed and may change during an experiment.
- The beam current is not measured via a charge collector but via a rotating vane system. This gives an energy spectrum that can be used to normalize the other spectra.

3.2.2 Monitoring demand

Three types of measurements are performed during a channeling experiment session:

- Orientation and calibration
  Before the main experiments several rough spectra are measured to check the set-up. Those spectra are measured with all detectors (currently two). Apart from providing a general check on the system, this can also be used to perform an energy calibration of the detectors.
- Angular scans
  A scan of the sample is made by measuring energy spectra for a series of orientations of the sample. This is done to roughly scan the sample for the orientations of the crystal axes.
- Search for the sample orientation with the minimum yield
  Finally, the sample orientation where the yield of backscattered projectiles reaches a minimum is determined. In that case, the beam axis is the aligned with a crystal axis.

During all of these experiments spectra are recorded for each detector, including the detector of the rotating vane system. All spectra have to be monitored by Columbus. It should be possible to use the rotating vane spectrum to normalize all the other spectra. This is done by integrating a part of the rotating vane spectrum and dividing the yield of the other spectra by this integral. There are two ways to handle the integration of the rotating vane spectrum:

- Integration of the rotating vane spectrum by PhyDAS and storage of the integral as an additional parameter in all the other spectra files that are sent to the Alpha workstation. Then Columbus can immediately use this additional parameter from the spectra file to normalize the energy spectrum before displaying it. A disadvantage of this method is that PhyDAS has to be told what part of the rotating van spectrum has to be taken into account. This can only be changed afterwards is the rotating vane spectrum is also stored.
- Both the rotating vane spectrum and the other spectra are sent to the Alpha and Columbus integrates the rotating vane spectrum before normalizing the other spectra. In this case an additional feature has to be added to the monitoring module, *i.e.* an integration unit.

The integration unit will also be necessary to create plots, or so-called 'minimum plots' of the backscatter yield as a function of the sample orientation.
An automatic determination of the orientation with the minimum yield will not be performed by Columbus because direct interaction between Columbus and the PhyDAS system is only possible by using the serial terminal line that is too slow.

3.2.3 Off-line analysis demand

Off-line analysis programs are needed for *e.g.* the determination from the angular scan measurement of the orientation with the minimum yield. All programs must be able to use data from the data base in Columbus.

3.3 Ion scattering requirements

3.3.1 Data base requirements

Requirements for the data base are similar to those for the other set-ups. All information about the experiment should be stored, including the information about the multi-parameter experiment set-up. The expansion to a virtually unlimited number of coincident parameters instead of just two coincident parameters (energy- and position) demands a complete redesign of the data base model.

3.3.2 Monitoring requirements

Requirements for the monitoring module are:

- List mode data with up to perhaps even 15 different parameters has to be handled. A large number of parameters is necessary for *e.g.* a CERDA-TOF experiment with two position sensitive detectors (each 3 output parameters) and a timing parameter.
- Function editor and pre-processor for creating new quantities (derived parameters) from the (primary) parameters in the list mode data. To illustrate this, consider a TOF experiment where a/o the energy E and the flight time t of a particle are measured as parameters in a list. Instead of a plot of E vs. t, a plot of Et^2 vs. E will be more useful because the contributions from particles with different mass will be separated as straight horizontal lines instead of bended curves. Separation is thus easier and can be performed more accurate.
- Display of spectra of both primary and derived parameters.
- Display of scatterplots, *i.e.* a plot of the spectrum of two parameters versus each other.
- The number of conditions (*e.g.* a limitation on the values for a parameter) on a spectrum or a scatterplot must be virtually unlimited.
- Finally, when the list mode data has been sorted out into scatterplots, it should be possible to select areas in these scatterplots and project the contents of the contours onto the two parameter axes, thus creating conditional histograms.

3.3.3 Off-line analysis requirements

The off-line analysis should offer even more extensive processing of the list mode data. It depends on computer power, how much of the data processing can be done by the monitoring module while keeping up with the experiment, and how much has to be done separately by the off-line analysis module.
The monitoring is expected to give an on-line overview of the experiment while the off-line analysis will process the list mode data into final results.

3.4 Design of the monitoring module

The monitoring module has to give an overview of the experiment that is being performed. This is done via graphics that are built from data in the files that have been created by the PhyDAS system. When new data files are provided by PhyDAS the graphics are updated. The number of graphics should be virtually unlimited and the definition of each graphic is taken from the data base.

One experiment at a time is handled. This can be either a list mode experiment or a histogram mode experiment. Experiments producing both list mode data and histogram mode data are not yet supported by PhyDAS. So within Columbus we will also keep these two types of experiments separated.

3.4.1 Monitoring of list mode data

During a multi-parameter list mode experiment, a list of data is measured for each parameter. These are the primary parameters. Together with primary parameters several additional parameters like measuring time and beam charge may be provided. The list mode data has to be sorted out into graphics. Two types of graphics are available:

- Single parameter histogram.
  The list mode data for one parameter is sorted out into a histogram. It should also be possible to create a histogram of a function of the primary parameters. Furthermore it should be possible to set windows on (functions of) parameters so a conditional histogram is obtained. A schematic example of a single parameter histogram graphic is shown in figure 3.3.

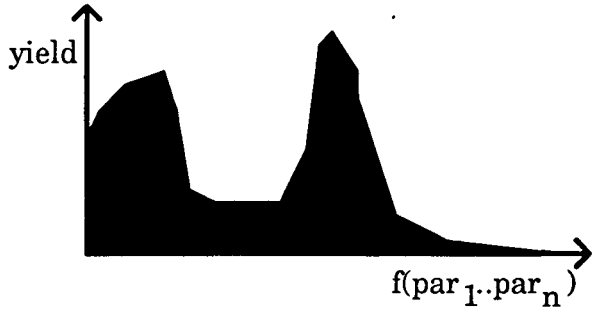

Figure 3.3: Schematic example of a single parameter histogram graphic.

- Two parameter scatterplot.
  In a two parameter scatterplot the histogram of two coincident parameters versus each other is shown. This can be primary or derived parameters and again it should be possible to set a number of conditions (windows) on parameters (both primary and derived) to obtain a conditional scatterplot. A schematic example of a scatterplot is given in figure 3.4.

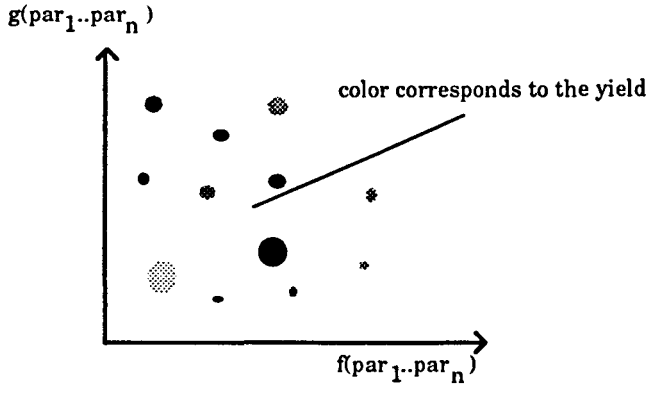

Figure 3.4: Schematic example of a two parameter scatterplot

Furthermore it should be possible to define a contour within a two parameter scatterplot and project the contents of this contour onto both axes into histograms. Finally, it should be possible to normalize the yield (for both types of graphics) on an additional parameter like the beam charge or the measuring time.

3.4.2 Monitoring of histogram mode data

When performing histogram mode experiments a spectrum file series for every parameter is created by PhyDAS. In addition to the histogram for one parameter, a file may contain additional parameters, like the measuring time, beam current, detector angle. These additional parameter are included in the file header.
Two types of graphics can be created:

- Single parameter histogram.
  The spectra from each file in the spectrum file series for a parameter are combined to a sum spectrum for that parameter. It should be possible to normalize the spectrum on either a additional parameter or on an integral of (a part of) the histogram of another. This histogram will also look like figure 3.3.
- Single parameter histogram versus additional parameter.
  In this case we do not create a sum spectrum from the individual spectra from the spectra series, but we take one additional parameter from the file header or an derived parameter, like the integral of a (part of) histogram from another parameter, and use this as the second parameter in the scatterplot. We then get a picture that looks like figure 3.4 again. Of course it should be possible to create the axis projection of the contents of a selected contour. (Example: when performing a channeling angular scan, a spectrum is measured for a number of angles of the goniometer. This is stored in one spectrum file series, where the angle is added to each file as the additional parameter. Creating a scatterplot of the energy spectra versus the angle, setting a contour on an energy interval and making an projection on the angle axis results in a 'minimum plot'.

All monitoring requirements for each set-up are fulfilled, when the design for the monitoring module, as described above, is implemented.

3.5 Design of the data base module

Changing Columbus into a multi-parameter data-acquisition package implies a complete revision of the data base. For every experiment, storage of the multi-parameter set-up is required, including coincident modules and a virtually unlimited number of parameters. Furthermore, the design of the monitoring module implies changes to the storage of the monitoring demand and the graphic definitions.

3.5.1 Data base model

A schematic overview of a data base design is given in a so-called data base model. A data base model consists of a number of entities (*e.g.* an experiment, a user, a sample and a parameter module). Each entity has its own table with corresponding attributes (*e.g.* the user table contains three attributes: an id., a name and a status). A table is a

collection of records (Each user is stored in an individual record). Every record consists of a number of fields that correspond to the attributes from the record table (The user record thus contains an id. field, a name field and a status field). Furthermore, a record has its own unique identification within a table through a key that consists of one or more fields (Every user gets a unique id.).
A table field can refer to a key field of another table (The experiment table has a field that refers to the id. of the user that performed the experiment). Thus a relation is formed between 2 entities. Three types of relations exist:

- *1:1 relation*
  Suppose the experiment entity has a 1:1 relation with the sample entity. This means that in each experiment only one sample is used and that each sample is used in only one experiment.
- *1:n relation*
  Suppose the user entity has a 1:n relation with the experiment entity. This means that an experiment is performed by one user, but this user can perform more than one experiment.
- *m:n relation*
  Suppose the experiment entity has a m:n relation with the module entity. This means that an experiment uses several modules and that a module is used in several experiments.

Figure 3.5 gives an overview of the new data base model for Columbus 2. It shows the entity tables together with their relations.
The central entity is the experiment entity (Experiment table). This can be either a main experiment, a focusing experiment or a calibration experiment. A main experiment has a reference to a focusing and a calibration experiment. A calibration experiment has a reference to a focusing experiment. Furthermore, each experiment has a reference to the experiment set-up (Configuration table) that was used in the experiment, to the user (Users table) that performed the experiment, to the sample (Sample table) that was used, to a scan pattern (Scanpattern table) if it was used and to a demand for monitoring (Monitoring table).
The sample entity refers to a client (Client table) from which the sample was obtained and to a description of the analysis (Analysis_demand table) that has to be done. Photos can be taken from a sample, thus the photo entity (Photo table) refers to the sample.
In the set-up entity (Configuration table) the modules (Module table) of which the set-up consists, can be selected. Because an experiment set-up can consists of several modules and a module definition can be used in several set-ups a cross reference is added (Module_in_Config cross-ref table).
In the monitoring demand the graphics (Graphic table) can be selected that have to be displayed during monitoring. Again because several graphics can be selected in the monitoring demand and because a graphic definition can be used in several monitoring demands a cross reference is added (Graph_in_Mon cross-ref table). A graphic refers to a parameter module in order to know what parameters (Parameter table) can be used.

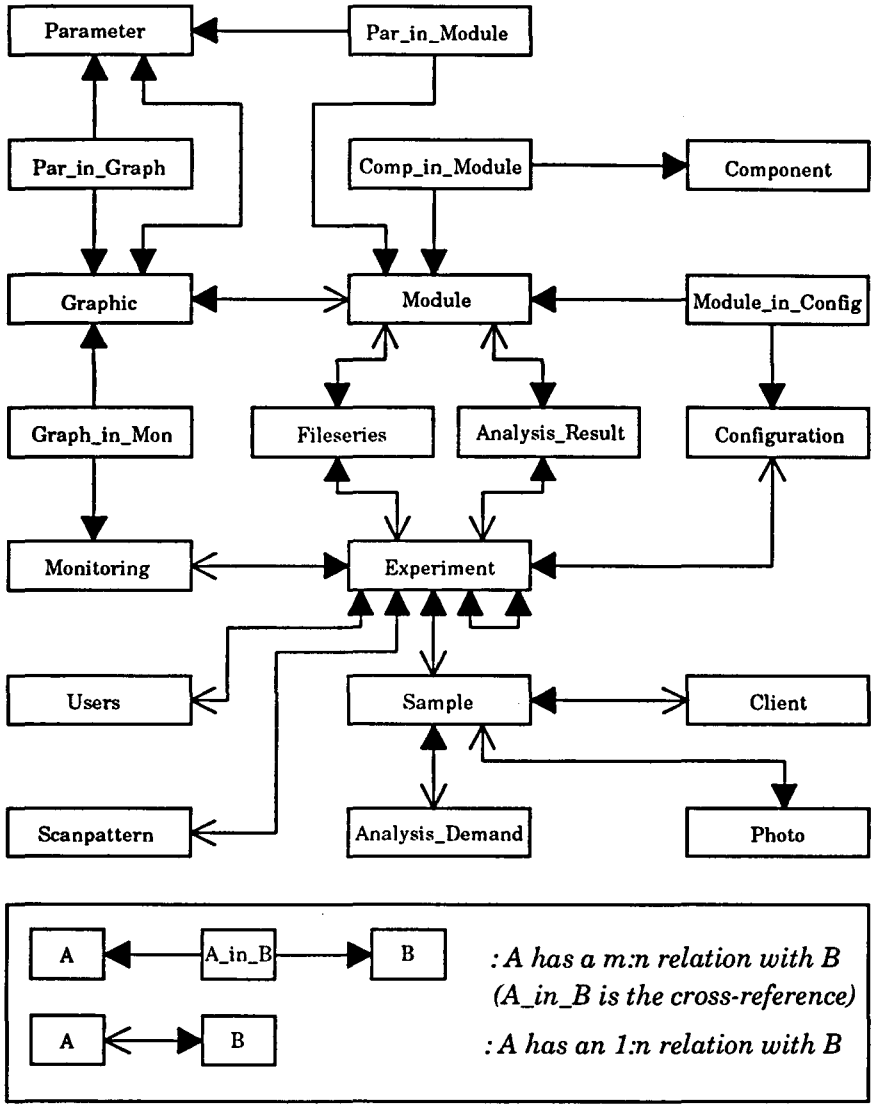

Figure 3.5: The new data base model

The graphic entity relates directly to the parameter entity for setting the parameters along the axes. Via a cross reference (Par_in_Graph cross-ref table) the conditions/windows on the parameters are stored.
The components (Component table) from which a module is built are also stored via a cross reference (Comp_in_Module cross-ref table).
A file series (Fileseries table) points to the experiment and module to which the data in the files belongs. Similarly, analysis results (Analysis_Result table) point to the experiment on which the analysis was performed and to the module from which the data came.
A full discussion of the new data base entities is given in appendix G.

3.6 Redesign of the user interface

A new data base module with new entities and new data fields also requires a redesign of the user interface for editing the data base. The functionality of the user interface is

kept the same as in the previous version. The new lay-out for every entity dialog is shown in appendix G together with the description of these entities.

3.7 Implementation of Columbus version 2

3.7.1 Implementation of the new data base model

The data base of Columbus 1 was a simple ASCII data base where the records were saved in a ASCII text files without any structures. When the amount of data that was stored in the data base became large, data base actions like storage and retrieval would slow down rapidly. Furthermore, no consistency check, *i.e.* checking if the contents of the data base are correct and valid, was performed at all. Although this data base was originally implemented as a temporary version it has never been replaced.
Because the data base module had to be rewritten anyway to make it support the new data base model, it was decided to replace the text based data base by the commercial data base Oracle. A complete description of the implementation of the Oracle data base and the integration in Columbus is given in reference [GER94].

3.7.2 Implementation of the new graphic user interface

In addition to the changes in the data base model itself, the graphic user interface of Columbus that is used to edit the data base entities was modified. The user interface of Columbus 1 was written in X Windows using the DEC Windows Toolkit (DWT), which is commonly used on DEC VAX stations. The 'look and feel' of DWT, *i.e.* the way the user interface looks and responds to the user, is flat and simple in contrast to the Motif Toolkit, that is available on the DEC Alpha station. The 'look and feel' of this windowing system is more fancy and up-to-date and furthermore, Motif has become a standard for X Windows systems. So the complete graphic user interface for the new version of Columbus has been rewritten using the X Windows Motif Toolkit. A capture of the main window of Columbus 2 is given in figure 3.6.

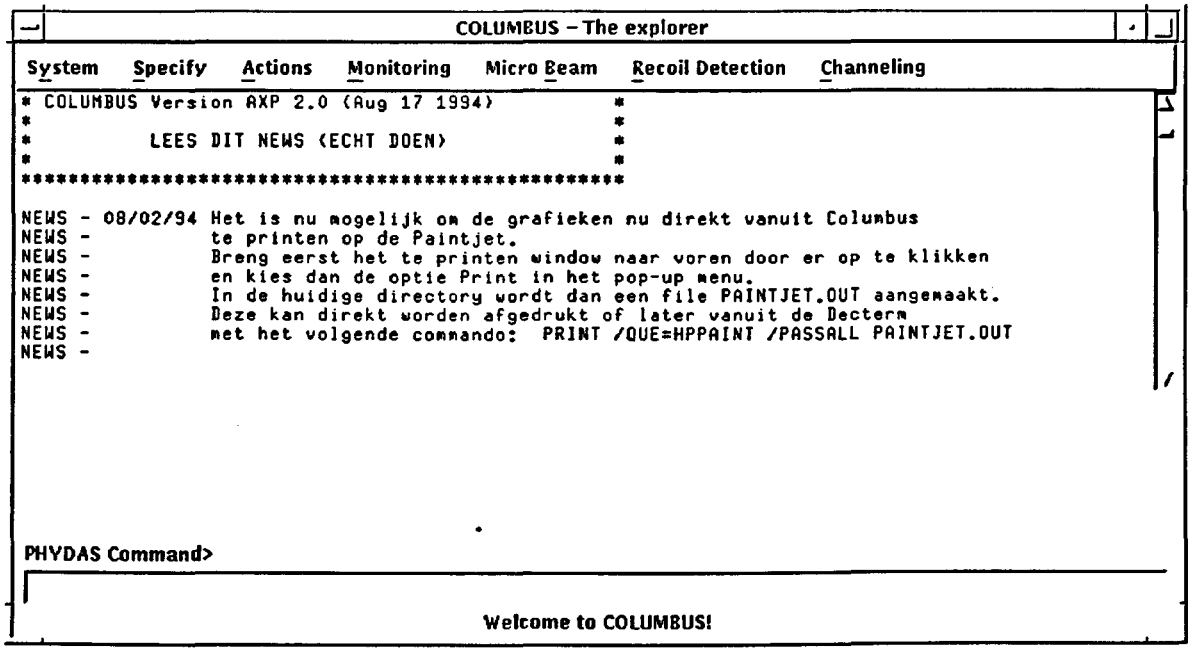

Figure 3.6: The main window of Columbus 2.
The Graphic User Interface (GUI) is created with the X Windows Motif Toolkit.

An overview of the user interface dialogs to edit the data base entities is given in appendix G together with the description of every data base entity.

3.7.3 Implementation of the multi-parameter monitoring module

A list of the accomplished implementations of the design for the new monitoring module (see section 3.5) is given below.

- Multi-parameter list mode experiments can be monitored. After selecting one of the experiments from the data base, the data from the corresponding list mode file series are monitored.
- The list mode data can be sorted out into a one parameter histogram and into two parameter scatterplots. The graphic definitions are taken from the data base. The parameters that are used in a scatterplot should be coincident and must thus in the same parameter module.
- In a two parameter scatterplot a polygonal contour, that indicates a region of interest, can be defined by using the mouse to indicate the corners of the contour. The contents of this contour can then be projected onto both the x-axis and the y-axis into conditional single parameter spectra.
- In data base graphic definitions a virtually unlimited number of conditions (channel windows) can be set on the graphics module parameters. An event from the list is only used to update a graphic if all conditions are satisfied.
- General features like resizing graphics, expanding regions of a graphic, linear or logarithmic yield scales, hardcopies of a graphic to the Paintjet colour printer or to a .PPM file have been implemented.
- The number of graphics is virtually unlimited. The only limitation is the amount of RAM memory that is available for Columbus. The size of a graphic in memory varies from about 10 kbyte for a one parameter histogram up to about 3 Mbyte for a two parameter scatterplot.

3.8 Suggestions for future work

A part of the design has not yet been implemented. A list of suggestions for future work is given below:

- Monitoring histogram mode data has to be added to the monitoring module, in addition to monitoring list mode data. This will require an expansion of the input routines that retrieve the data from the files. These input routines will have to support both list mode data files and histogram mode data files. Only minor changes have to be made to the routines for drawing graphics and to the data base module. When histogram mode monitoring has been implemented, Columbus can be used to monitor histogram mode data from for example channeling experiments.
- List mode monitoring has to be expanded with an equation interpreter that will process user defined equations of experiment parameters into derived parameters. This is needed for *e.g.* microbeam experiments where the PIN parameter from the scan pattern generator has to be converted into a horizontal and vertical position co-ordinate for creating distribution plots.

In a time-of-flight experiment, the energy and flight time of a particle are measured as experiment parameters and the mass of the particle can be derived from these experiment parameters.
The data base model has to be expanded with an entity for storing the equations. Experiment parameters that are used in an equation have to be coincident to allow calculation of a derived parameter for every event. Therefore the equation entity has to refer to a module entity, to know what parameters are available. A cross-reference entity may be needed for keeping track of the parameters that are used in an equation.

- Although it is possible to store graphics to disk (either as a bitmap or as raw ASCII data), it is not yet possible to reload a graphic in Columbus. A general file format for storing Columbus graphics has to be designed and an input routine has to be implemented that can retrieve the graphic files. Other applications, that may have processed or generated graphics, can store their output in the Columbus graphic format and Columbus can be used to display the graphics.

Chapter 4
Application:
Multi-Parameter Pulse Shape Discrimination

Scattering experiments using the Elastic Recoil Detection Analysis with Pulse Shape Discrimination (ERDA-PSD) technique were performed to show the increased possibilities of the multi-parameter data-acquisition system compared to the conventional single parameter data-acquisition. In section 4.1 the theory of Elastic Recoil Detection Analysis is discussed, followed by the Pulse Shape Discrimination theory in section 4.2. In section 4.3 the electronics that was used in the set-up is described. Pulse rise-time discrimination measurements and improvements are discussed in section 4.4 and section 4.5. The same is done for pulse height measurements in sections 4.6 and 4.7. Some general improvements are discussed in section 4.8 followed by encountered problems and effects in section 4.9.

4.1 ERDA theory

Section 4.4.1 describes the kinematics of a binary collision. In section 4.1.2 the energy loss of ions moving through matter is discussed. In section 4.1.3 the probability of an ion to be scattered into a certain solid angle is discussed.

4.1.1 Binary collision kinematics

The kinematics of a binary collision between an incoming projectile and a target atom at rest can be derived from the laws of energy and momentum conservation. A difference has to be made between elastic and inelastic collisions. During an inelastic collision an amount of energy Q is transferred from the projectile nucleus to the target nucleus. This amount is defined to be positive and can be set to zero for elastic collisions.

Figure 4.1 shows the laboratory frame geometry of a binary collision between an incoming projectile with mass m_1 and initial energy E_0 and a target atom at rest with mass m_2. After the collision the projectile scatters into an angle θ with energy E_1. The target atom recoils into an angle φ with energy E_2. The mass ratio μ is defined as $\mu = m_2/m_1$.

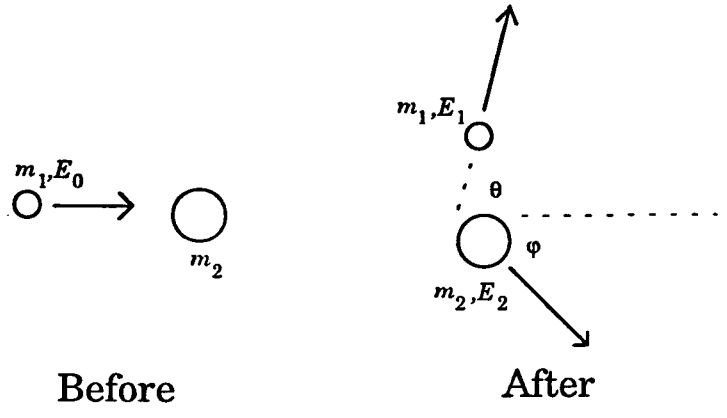

Figure 4.1: The laboratory frame geometry of a binary collision

The energy of both the scattered projectile and the recoiled particle after the collision can be calculated with the kinematic relations that are given below [DIJ92].
In equation 4.1 the kinematic factor K_{scat} for a scattered projectile is defined.

$$K_{scat} \equiv \frac{E_1}{E_0} = \left(\frac{\cos\theta + \sqrt{\mu^2\left(1 - \frac{Q}{E_0}\left(\frac{\mu+1}{\mu}\right)\right) - \sin^2\theta}}{\mu+1} \right)^2 \tag{4.1}$$

A second solution exists when $\sin^2\theta < \mu^2\left(1 - \frac{Q}{E_0}\left(\frac{\mu+1}{\mu}\right)\right) < 1$:

$$K_{scat} \equiv \frac{E_1}{E_0} = \left(\frac{\cos\theta - \sqrt{\mu^2\left(1 - \frac{Q}{E_0}\left(\frac{\mu+1}{\mu}\right)\right) - \sin^2\theta}}{\mu+1} \right)^2 \tag{4.2}$$

The kinematic factor K_{rec} for the recoiled particle is defined in equation 4.3.

$$K_{rec} \equiv \frac{E_2}{E_0} = \frac{\mu}{(\mu+1)^2}\cos^2\varphi\left(1 \pm \sqrt{1 - \frac{(\mu+1)Q}{\mu E_0 \cos^2\varphi}}\right)^2 \tag{4.3}$$

From energy conservation follows relation 4.4 between K_{rec} and K_{scat}:

$$K_{rec} = 1 - K_{scat} - \frac{Q}{E_0} \tag{4.4}$$

4.1.2 Stopping power

The energy of the scattered projectiles and recoils does not solely depend on the kinematics of the scattering process. When particles move through matter they lose energy because of interaction with the electrons and atoms they encounter. This effect is called (electronic and nuclear) stopping. The stopping power, or specific energy loss, is defined as the differential energy loss divided by the differential path length:

$$S \equiv -\left(\frac{dE}{dx}\right) \tag{4.5}$$

A stopping power curve shows the stopping as a function of the energy of the moving particles and a Bragg curve shows the stopping as a function of the depth of a moving particle. Examples of these curves are given in figure 4.2.

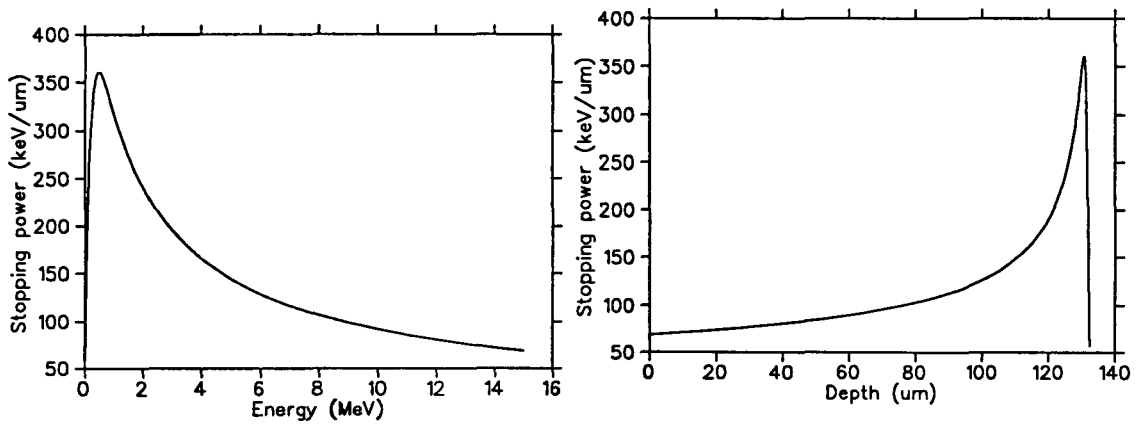

Figure 4.2: Stopping power as a function of energy for alphas in Si (left) and stopping power as a function of depth for 11.0 MeV alphas in Si (right).

Stopping by interaction with electrons dominates if the particle velocity is large compared to the orbital velocity of the electrons in the absorbing atoms. Electronic stopping decreases with increasing velocity. This can be understood by noting that at higher velocities the particle spends less time in the vicinity of any electron and less energy is transferred. Electronic stopping roughly varies inversely with the particle energy. Furthermore, the stopping varies with the mass of the moving particle and the square of its charge. Particles with the greatest charge and mass will have the largest specific energy loss.

At low particle velocities, nuclear stopping through interaction of the moving ion with the target atoms as a whole dominates. In this velocity region, below the stopping power maximum, stopping is proportional to the particle velocity.

The Bragg curve shows that high energetic particles deposit most of their energy at the end of their track through matter.

The energy $E(d)$ of a particle with an initial energy $E(0)$ that moves through matter over a distance d can be calculated with equation 4.6: [FEL86]

$$E(d) = E(0) - \int_0^d S(E(x))\mathrm{d}x \tag{4.6}$$

If the stopping power is fairly constant over the path length d, the integral in equation 4.6 can be approximated by $S \cdot d$, where S is evaluated at some average energy between $E(0)$ and $E(d)$.

4.1.3 Cross sections

The number of target atoms per unit area, N_S, determines the probability of a collision between the incident particles and the target atoms. This probability is measured by the total number is detected particles Q_d for a given number Q of incident particles. The connection between N_S and Q_d is given by the scattering cross section.

The differential recoiling cross section $[\mathrm{d}\sigma/\mathrm{d}\Omega]_{rec}$, of a target atom to recoil into a differential solid angle $\mathrm{d}\Omega$ centered about φ is given by equation 4.7. [FEL86]

$$\frac{d\sigma(\varphi)}{d\Omega}\cdot d\Omega\cdot N_s = \frac{\text{Number of particles scattered into } d\Omega}{\text{Total number of incident particles}} \tag{4.7}$$

When the detector solid angle Ω is small ($<10^{-2}$ sr), the scattering cross section $\sigma(\varphi)$ can be defined as:

$$\sigma(\varphi) = \frac{1}{\Omega}\int_{\Omega}\frac{d\sigma}{d\Omega}\cdot d\Omega \tag{4.8}$$

An approximation for the solid angle Ω of a (small) detector is given by $\Omega = A\cdot l^{-2}$ [sr], where A [m^2] is the area of the detector and l [m] the distance at which the detector is located.
The number N_S of target atoms per cm^2 is related to the yield of detected particles by:

$$Y = Q_d = \sigma(\varphi)\cdot\Omega\cdot Q\cdot N_s \tag{4.9}$$

When the interaction between the projectile and the target atom is only through Coulomb forces, the differential recoiling cross section $[d\sigma/d\Omega]_{rec}$ can be calculated with the Rutherford formula for recoils (given in the laboratory frame of reference) [JAE94]:

$$\left[\frac{d\sigma}{d\Omega}\right]_{rec} = \left(\frac{Z_1 Z_2 e^2}{8\pi\varepsilon_0 E}\right)^2\left(\frac{\mu+1}{\mu}\right)^2\frac{1}{\cos^3\varphi} \tag{4.10}$$

where Z_1 and Z_2 are the nuclear charges of the projectile and the target atom, respectively, and where E is the energy of the projectile.
For higher projectile energies the repulsive Coulomb barrier is overcome and nuclear interactions become important. The cross section then deviates from the Rutherford value and depends strongly on both energy and angle. The energy E_{nr} above which this deviation occurs can be estimated with equation 4.11 in which the energy is calculated for which the distance of closest approach is comparable to the nuclear radius of the target atom.

$$E_{nr} = \frac{Z_1 Z_2 e^2}{R_0 A^{1/3}} \tag{4.11}$$

with Z_1 and Z_2 the atomic numbers of the projectile and the target atom, respectively, A the mass number of the target atom and a characteristic radius $R_0 \cong 1.4\times10^{-13}$ cm.

4.2 PSD theory

Essential for ERDA is the discrimination between scattered projectiles and recoils. For this, one can use the difference in stopping power between scattered particles and recoils. The range of alpha particles is much larger than the range of recoils (like carbon, oxygen and nitrogen ions) with the same energy. The range of 8 MeV alphas in Si is 48 μm and the range of carbon recoils with the same energy is about 7 μm. An extremely thin detector can be used to separate the long range particles from the short range

recoils. Thin detectors are, however, expensive because they need special fabrication processes.
Instead, a low resistivity Si semi-conductor detector at low bias can be used. The thickness δ of the depletion layer (in μm) can be calculated using equation 4.12 [RIJ93]:

$$\delta \approx 0.56\sqrt{\rho(V_b + 0.6)} \qquad (4.12)$$

where ρ is the bulk resistivity (in Ωcm) and V_b the applied external bias voltage (in Volts). For a detector with ρ=500 Ωcm and no external bias voltage the thickness of the depletion layer will be about 10 μm. This thickness may be increased by applying an external bias voltage.
A depletion layer thickness of 10-20 μm is large enough to stop all recoils (when using the current cyclotron). The charge carriers that the recoils liberate within the depletion layer are collected quickly and the charge collection times are in the order of nanoseconds [RIJ93]. These times can be neglected compared to the electronic rise times in our set-up.
Alpha particles on the other hand, pass through the depletion layer and are stopped in the neutral zone behind the depletion layer. They will only deposit a part of their energy within the depletion layer and the rest is used to liberate charge carriers behind the depletion layer. In addition to the fast collection of charge carriers from within the depletion layer, a part of the charge carriers in the neutral zone will be collected by the process of diffusion. Charge collection from layers just behind the depletion layer is also fast, thus increasing the effective depletion layer thickness with a few μm [RIJ93], but charge collection by diffusion from deeper layers is rather slow. This adds a slow component to the output pulse of the detector and decreases the rise time of this pulse.
By analysing the rise-time of the detector charge pulse, a particle that produces a pulse with a slow component (long range alpha particles) can be discriminated from particles that give a fast pulse (recoils and slow alpha particles). This technique is called Pulse Shape Discrimination (PSD).
Of course, in addition to the rise-time analysis of the detector charge pulse for discrimination purposes, the pulse height has to be analysed for the actual energy measurements. This leads to an ambiguity, because fast timing measurements require a high pulse *slope*-to-noise ratio, which is obtained if the time constant of a pulse shaping circuit is equal to the charge collection time, whereas energy measurements require a high pulse *height*-to-noise ratio, which is realized by a pulse shaping circuit with time constants of about 1 μs.
The ambiguity can be resolved by using a pre-amplifier unit containing both a fast voltage sensitive amplifier and a charge sensitive amplifier. A detector voltage pulse with a rise time t_c that appears at the input of the pre-amplifier, will first cause an output pulse at the timing output from the fast voltage sensitive amplifier with a rise time that is comparable to t_c. At the energy output of the charge sensitive amplifier an output pulse follows with a rise time that is generally much larger than t_c and proportional to the detector capacity C_d [RIJ93].

The energy output signal from the pre-amplifier is amplified and shaped by a main amplifier to increase to signal-to-noise ratio and to obtain a pulse with a shape that is suitable for further processing. (See *e.g.* [KNO79]).
The fast timing output and the energy output are connected to an additional electronic circuit. In conventional pulse shape discrimination, the electronic circuit is used for both analysis *and* discrimination. Only an alpha-suppressed energy spectrum is recorded [RIJ93].
Instead, for multi parameter PSD, an electronic circuit is used that *only* analyses the fast timing output and the energy output and that gives a number of output signals that contain information about the energy of the particle and the rise-time of the detector output pulse. The exact composition this electronic circuit is given in section 4.3.
Discrimination is now done by a multi parameter data-acquisition system, that records *all* output signals and allows a discrimination between the alphas and the recoils that may be more flexible and accurate than discrimination by an electronic circuit.
Two ways of pulse shape analysis that will allow discrimination with a multi parameter data-acquisition system are discussed: section 4.2.1 describes pulse rise-time analysis and section 4.2.2 describes pulse height analysis.

4.2.1 Pulse rise-time analysis

The energy output of the charge sensitive pre-amplifier is amplified by a main amplifier. The amplifier contains two shaping filters. One is a CR-RC shaping filter which is a combination of a differentiator and an integrator with an equal time constant τ_E=RC. With this filter, a unipolar pulse is obtained that is used for energy measurement. Furthermore, a CR-RC-CR shaping filter, with an extra differentiator, gives a bipolar output pulse with a zero-crossing.
Both the rise-time of the unipolar output pulse and the zero-crossing time of the bipolar pulse depend on the speed of charge collection in the detector, and thus on the rise-time of the detector charge pulse.
Rise times of detector pulses from recoils and short range alpha particles are equal and the corresponding bipolar pulses cross zero at the same time. However, bipolar pulses from alpha particles that are stopped behind the depletion layer and for which the charge collection is slow due to diffusion processes, cross zero at later times.
Differences in zero-crossing times are used to discriminate between the short-range particles and the long-range particles.

4.2.2 Pulse height analysis

If shaping times of shaping circuits become comparable with the rise time of the pulse from the pre-amplifier, its pulse shape is no longer a step function input to the shaping filter and some pulse height is lost. This loss is called the *ballistic deficit* [KNO79] and this is usually avoided by keeping the time constants long compared to the rise time of the input pulse that is to be shaped (Not too long, because this will increase pile-up!).
This effect can, however, be used to discriminate between fast and slow preamplifier pulses; the energy output from the pre-amplifier is split up and connected to two (or

more) main amplifiers with unipolar shaping filters that have different shaping time constants τ_{E1}, τ_{E2} *etc.* ranging from a few μs to less than a μs. These time constants are large compared to the rise time of fast preamplifier pulses from short range particles, like recoils. For these fast pulses no ballistic deficit occurs and the pulse heights will correspond to the particles energy.
Rise times of slow pulses from long range particles, however, are comparable to the time constants and a ballistic deficit occurs. This deficit differs for each shaping time.
With a multi-parameter data-acquisition system, this pulse shape dependant response of the shaping filters can be used to discriminate between the fast and the slow detector output pulses of the recoils and projectiles, respectively.

4.3 Experiment set-up

A Passivated Implanted Silicon Planar (PIPS) detector with a bulk resistivity ρ = 500 Ω cm and good timing properties is used as the ERDA-PSD detector. A 1 mm vertical slit was placed in front of the ERDA-PSD detector to reduce the angular range and thus the kinematic spread of the detected particles.
An extra detector was positioned at a backward angle to monitor the beam current.
Section 4.3.1 describes the electronics used for the PSD analysis. Section 4.3.2 describes the electronics for the monitor signal.

4.3.1 The PSD electronics

The ERDA-PSD detector is connected to an electronic circuit that will allow both pulse height analysis and pulse rise-time analysis. The output signals (parameters) of the PSD electronic circuit are connected to the multi-parameter data-acquisition system. In figure 4.3 an overview of the PSD electronics is shown.

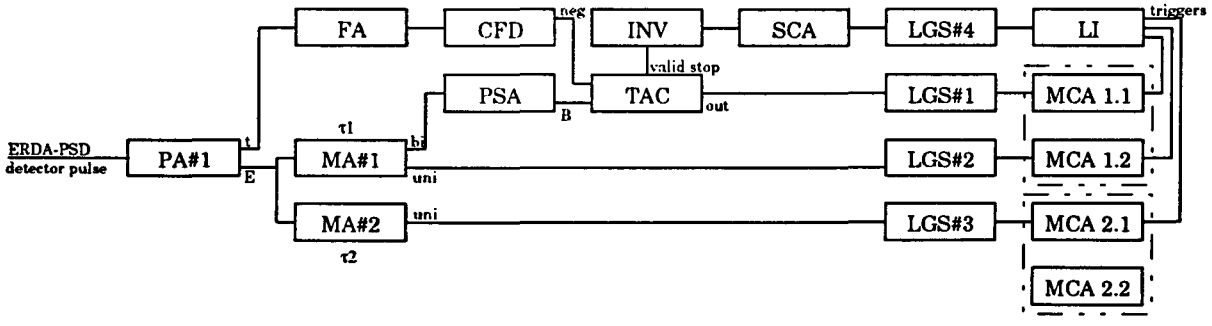

Figure 4.3: An overview of the Pulse Shape Discrimination electronics.
(PA = Pre-amplifier, MA = main amplifier, FA = fast amplifier,
CFD = constant fraction discriminator, PSA = pulse shape analyser,
TAC = time-to-amplitude converter, INV = inverter,
SCA = single channel analyser, LGS = linear gate stretcher,
LI = linear interface, MCA = multi channel analyser)

The ERDA-PSD detector is connected to a pre-amplifier (PA#1, Silena Catsa 82). The energy output of this pre-amplifier is connected to two main-amplifiers (MA#1, Canberra 2020 and MA#2, Ortec 572) with different shaping times τ_1 and τ_2, respectively. Typical values are τ_1=0.25 μs and τ_2=1.0 μs. The unipolar output of each

main amplifier is connected to a stretcher (LGS#2 and LGS#3, respectively) of which the output is connected to the ADC input of a MCA (MCA 1.2 and MCA 2.1, respectively). These are the two energy parameters $E_{\tau 1}$ and $E_{\tau 2}$.
In addition to the energy output, the pre-amplifier (PA#1) has a fast timing output; the pulse from that output is amplified by a fast amplifier (FA, Ortec VT120) and led to a constant fraction discriminator (CFD, Canberra 2126) that will generate a logic negative output pulse whenever the height of the input signal is at a constant fraction of its maximum. This output pulse is used as the start pulse for the time-to-amplitude converter (TAC, Canberra 2145) that converts the time difference between a start signal pulse and a stop signal pulse into a pulse with a height that corresponds to the time difference.
The bipolar output of main-amplifier MA#1 is connected to the pulse shape analyser (PSA, Ortec 552) that will generate a pulse on the zero crossing of the input signal. This pulse is used as the stop pulse for the TAC. The TAC output signal is connected via a stretcher (LGS#1) to the ADC of MCA 1.1. This is the timing parameter t.
The ADCs of the MCA boards each need a trigger pulse that is created as follows. The TAC valid stop gate output signal is inverted using an inverter unit (INV, Ortec 433A). This inverter is connected to a single channel analyser (SCA, Ortec 550A) that is used to generate a trigger pulse for every inverted gate signal. The trigger signal is split up with a linear interface (LI) and is connected to the ADC trigger inputs of the MCA 1.1, 1.2 and 2.1.
The stretchers are used to stretch and delay all signals so they arrive at the same time at the ADCs. Thus only one trigger signal has to be created for all three coincident parameters (the timing parameter and the two energy parameters). The discrimination level of the stretchers should be sufficiently high to avoid 'noise stretching' instead of stretching the real signals.
By using one trigger for the three coincident signals, a coincidence is faked. New hardware is being developed that will allow real coincidence measurements. These new peak detector units will generate a trigger signal for each individual parameter. Whether signals are truly coincident will then be determined by a gate signal that is connected to each MCA. If the triggers for all coincident parameters fall within the gate signal the coincidence will be judged valid.

4.3.2 The monitor signal electronics

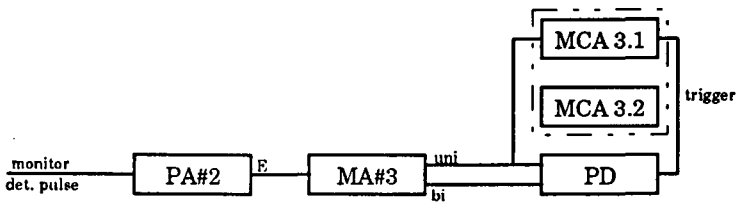

Figure 4.4: An overview of the monitor signal electronics.

The monitor detector signal is amplified using a second pre-amplifier (PA#2, Silena Catsa 82) and a main-amplifier (MA#3, Ortec 572) of which the unipolar and bipolar output signals are connected to a peak detector unit (PD) that will generate a trigger for the ADC of MCA 3.1 on the maximum of the unipolar signal. Of course, this unipolar signal is also connected to the signal input of MCA 3.1.

4.4 Pulse rise-time discrimination measurements

4.4.1 A thin carbon foil

A thin carbon foil (10 μg/cm^2) was bombarded with 13.4 MeV alpha particles at normal incidence (ψ=90°) and the recoils and scattered alphas were detected with the ERDA-PSD detector, operated at 1.0 V bias and located at an angle of φ=30°.

Figure 4.5 shows the timing *vs.* energy (E/t) scatterplot that was obtained by monitoring the list mode data with Columbus and plotting the timing parameter *vs.* one of the energy parameters.

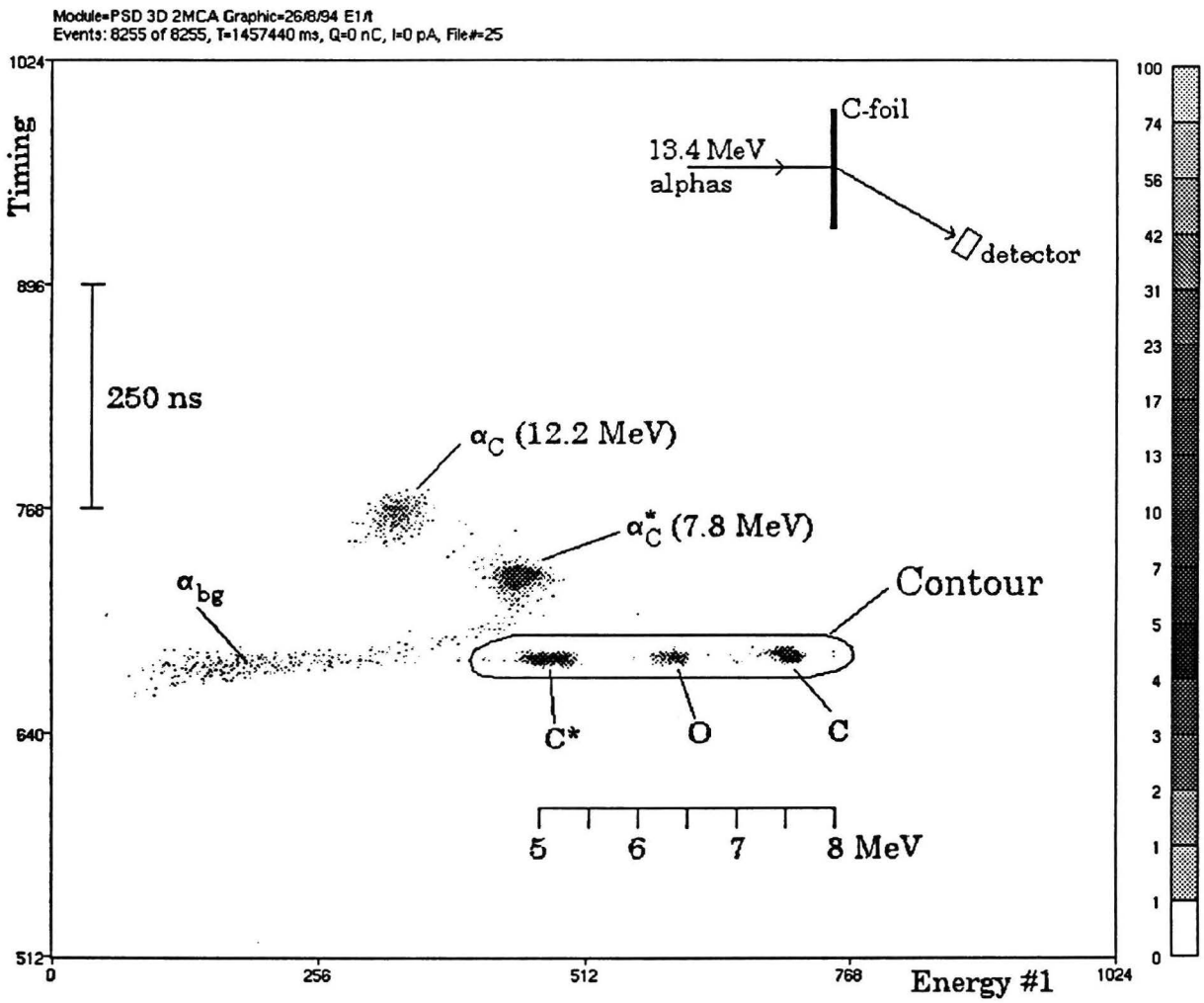

Figure 4.5: Timing vs. energy scatterplot of a 10 μ g/cm^2 carbon foil. (φ=30°, ψ=90°, E_0=13.4 MeV, V_b=1.0V, τ_t=0.25 μs, τ_{E1}=0.25 μs)

The energy parameter E_1 (shaping time τ_{E1} equal to 0.25 μs) is plotted along the horizontal axis and the timing parameter t (shaping time τ_t equal to 0.25 μs) along the vertical axis. Units are in channels; Columbus will soon support axis calibrations.
Particles that are stopped in the depletion layer, like the recoils, all have the same fast timing. Only their energy differs, so they will fall on a straight line in the E/t scatterplot. The elastic carbon recoils (C, 7.5 MeV), the inelastic carbon recoils (C^*, 5.0 MeV) and some elastic oxygen recoils (O, 6.4 MeV) due to surface contamination, indeed appear on this straight line. Although the surface contamination is small, it is visible because the cross section for oxygen recoils has a maximum at 30 degrees for the beam energy of 13.4 MeV [IJZ93].
Alpha particles that are scattered on carbon are stopped beyond the depletion layer and give a slower timing signal. Elastically scattered alphas on carbon (α_C, 12.2 MeV) and inelastically scattered alphas on carbon ($\alpha_C{}^*$, 7.8 MeV) indeed display a larger timing signal and appear separated from the recoils. Notice that the alphas appear with a lower energy value than the recoils and that the measured energy of the elastic alphas on carbon is lower than for the inelastic alphas on carbon. This is caused through charge collection that is slow or even incomplete. This will be discussed further in section 4.4.3.
A background of scattered alphas (indicated by α_{bg}) with all kinds of energies is also visible. (The cause of this background is discussed in section 4.9.1).
The contents of the scatterplot can be projected on both axes into histograms. This is done in figure 4.6 for the timing axis and in figure 4.7 for the energy axis.
As can be seen in figure 4.6 the timing peaks from the alphas appear separated from the recoil timing peak. In conventional PSD experiments, a single channel analyser is used to select the events with a timing signal within a region around the recoil timing peak.

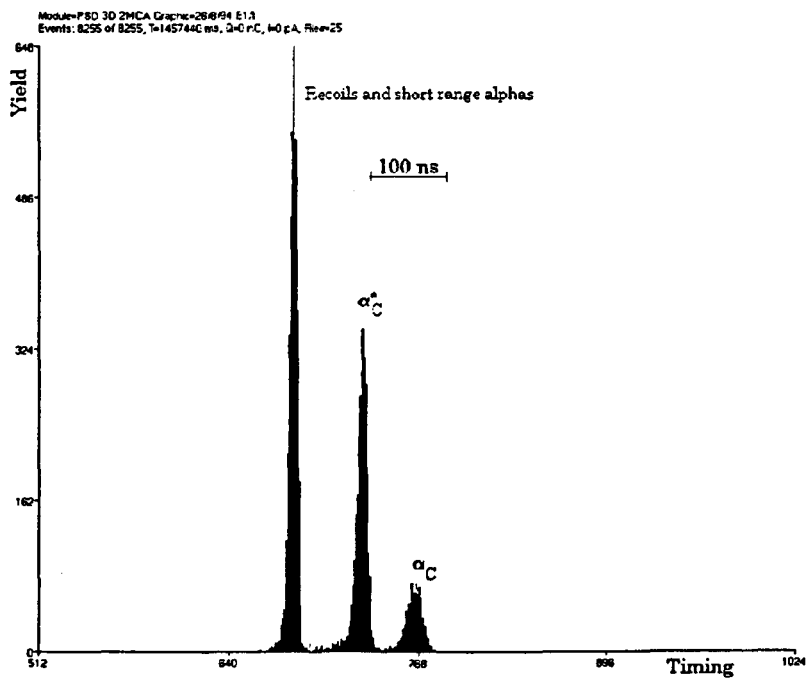

Figure 4.6: Timing axis projection of the scatterplot in figure 4.5.

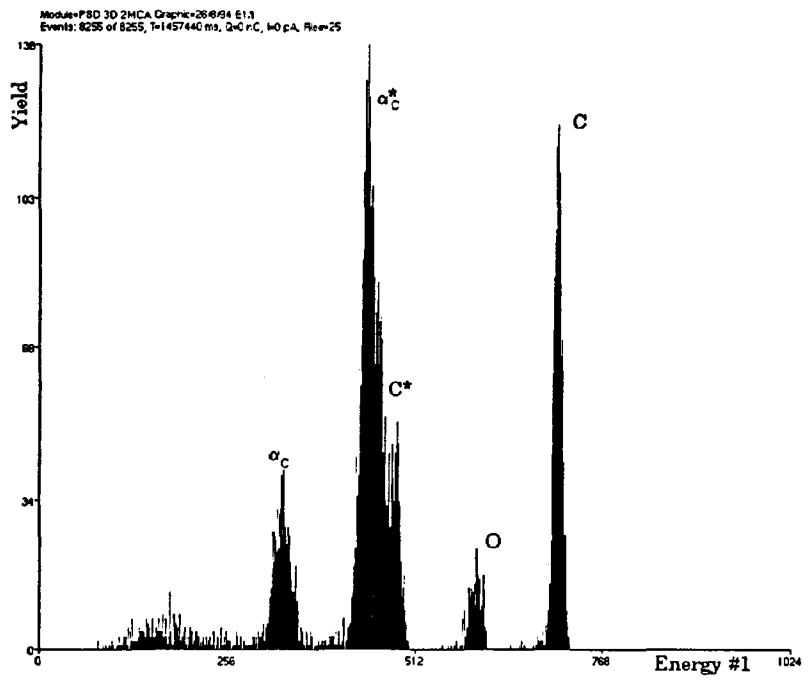

Figure 4.7: Energy axis projection of the scatterplot in figure 4.5.

The disadvantage of this method is that rejected events are thrown away and if the region is not selected correctly, the experiment has to be repeated with another region selection. For other samples, *e.g.* thicker or mounted differently in the target holder, the locations of the timing peaks may shift or broaden which also results in an improper discrimination.

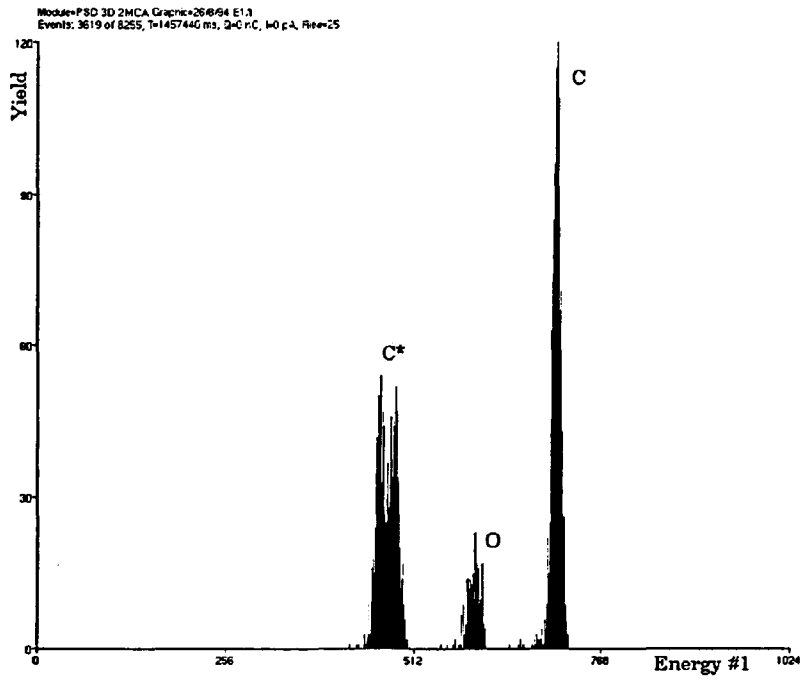

Figure 4.8: The energy histogram resulting from the projection of the contents of a contour enclosing the recoil contributions of the scatterplot 4.5 onto the energy axis.

With a multi-parameter data-acquisition system the discrimination can be repeated over and over. Data from all events is recorded and displayed in scatterplots like figure 4.5 or in histograms. From the timing parameter histogram a the region of interest that encloses the recoil events can be determined. Then an energy histogram can be generated, applying the determined conditions on the timing parameter.
An even more powerful tool is the possibility to select all recoil events by drawing a contour in the E/t scatterplot around the recoil contributions. The contents of this contour can be projected onto the energy axis and the result is a alpha-suppressed energy histogram, as shown in figure 4.8.
A 'clean' energy spectrum of the recoils without an alpha background is obtained. Energy spectra of thin carbon foils are used for energy calibration of the detector when the beam energy and the detection angle are known. On the other hand, measurements of energy spectra like these at several detection angles can be used to determine the beam energy and the deviation of actual the beam entrance angle from the expected entrance angle, by fitting the measured spectra with the kinematic relations.

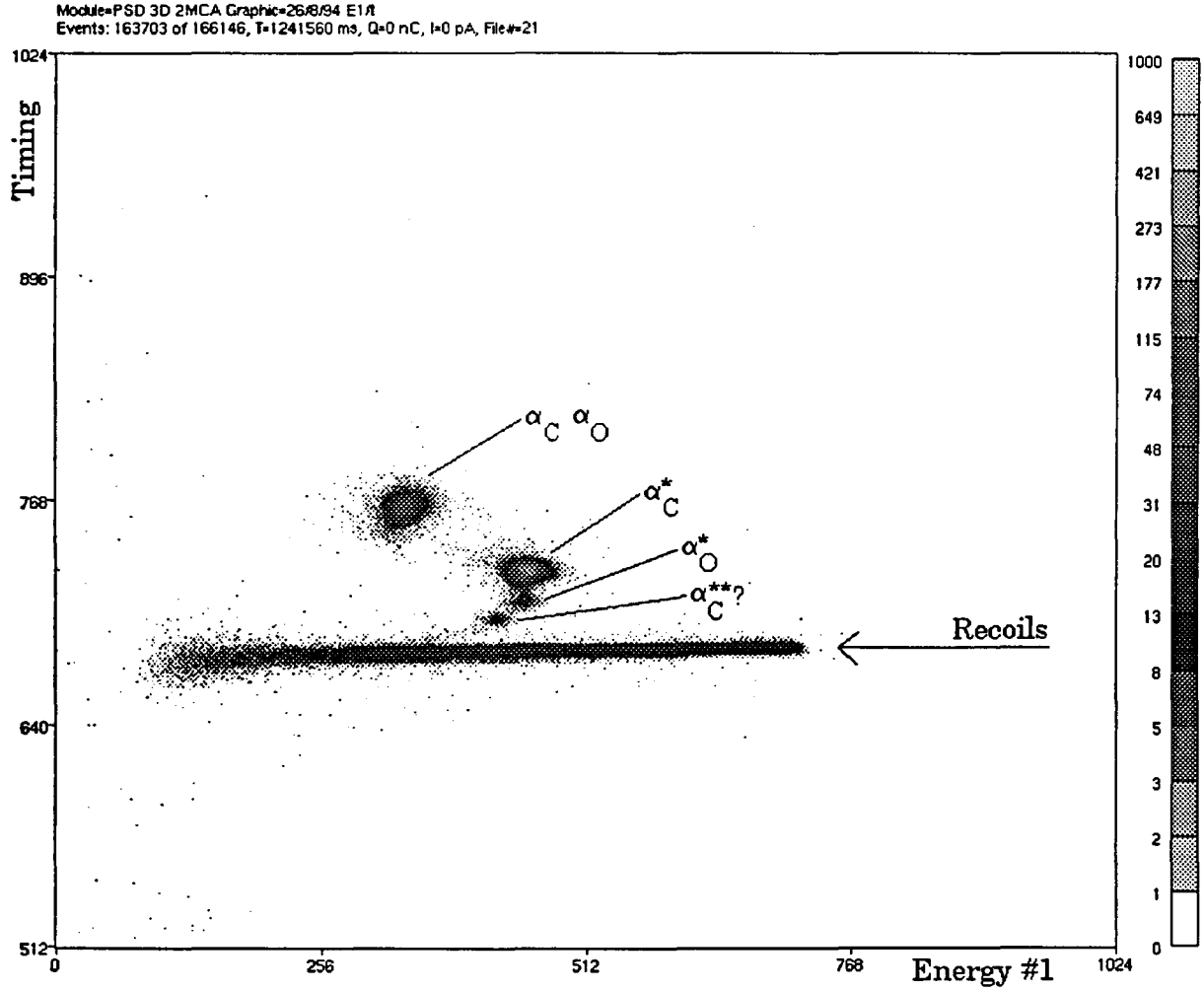

Figure 4.9: The E/t scatterplot of a Mylar foil
(φ =30° , ψ =90° , E_0=13.4 MeV, V_b=1.0V, τ_t=0.25 μ s, τ_E=0.25 μ s)

4.4.2 A Mylar foil

A mylar foil contains, in addition to carbon, also oxygen and hydrogen. Figure 4.9 shows the E/t scatterplot of a mylar foil that was bombarded with 13.4 MeV alphas at normal incidence. The detection angle was 30 degrees.

The oxygen alphas are now more visible in the scatterplot. Because this Mylar foil (thickness about 4.2 μm) is thicker than the carbon foil the peaks are broadened and start to overlap for the recoils. This can also be seen in figure 4.10, where the corresponding alpha suppressed energy histogram is shown (only contribution to the recoil line in figure 4.9 were used to create the histogram).

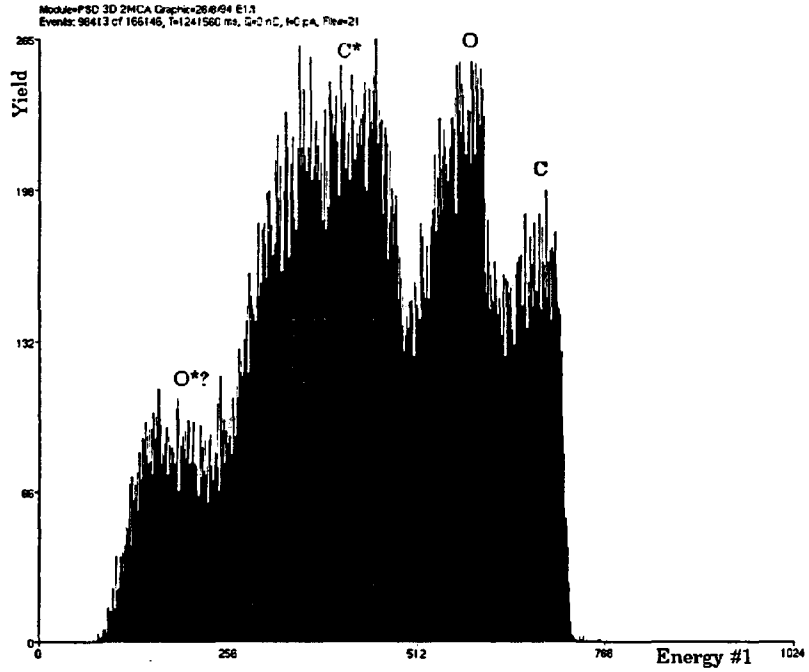

Figure 4.10: Energy histogram of the recoils from fig. 4.9

4.4.3 A Si_2O_3N layer on a Si substrate

As an example of a thick sample, the measurement on a 80 nm Si_2O_3N layer deposited on a Si substrate is discussed. Figure 4.11 shows the E/t scatterplot.

The recoils of course appear on the straight line: some elastic carbon recoils (C, 7.5 MeV) due to contamination and many oxygen recoils (O, 6.4 MeV) are clearly visible. The contributions from nitrogen recoils (N, 6.95 MeV) and the inelastic carbon recoils (C*, 5.0 MeV) are less visible. The cross-section for nitrogen is low for this angle and energy. [IJZ93]. To see more nitrogen, one should measure at another angle or energy where the cross section of nitrogen is large compared to the cross-section of oxygen.

When bombarding a thick sample, alphas are detected with a wide range of energies corresponding to all kinds of depths at which they were scattered. This range of alpha energies is visible as an 'alpha curve' in the E/t scatterplot.

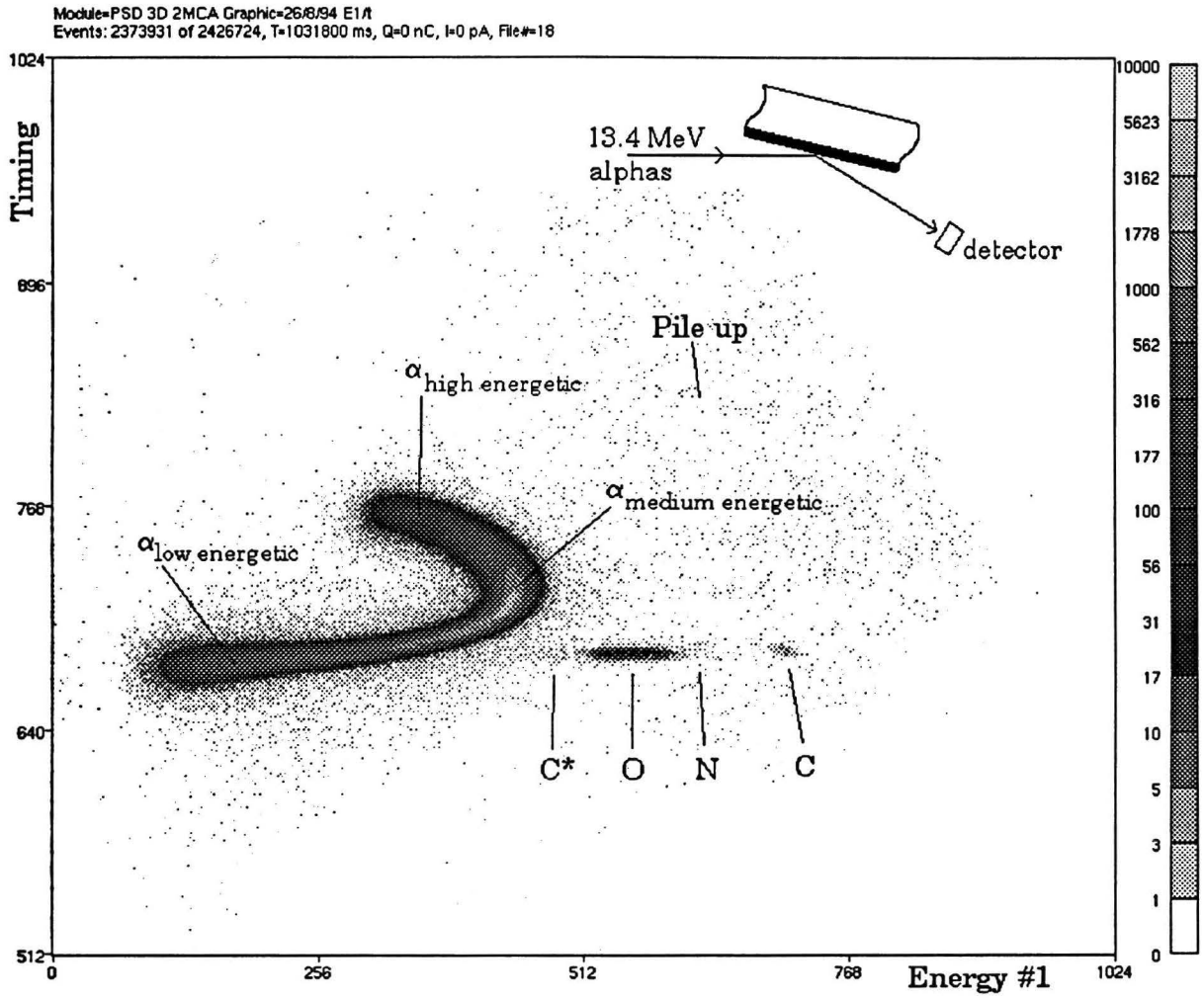

Figure 4.11: The E/t scatterplot of a Si_2O_3N layer on a Si substrate.
(φ =30° , ψ =15° , E_0=13.4 MeV, V_b=1.0V, τ_t=0.25 μ s, τ_E=0.25 μ s)

Low energetic, short range alpha particles ($\alpha_{low\ energetic}$) have fast timing signals because they are stopped within the depletion layer or just behind it. Their timing signals are comparable to the recoil timing signals and they form the low energetic tail of the alpha curve.

Alpha particles with an energy comparable to the recoil energies ($\alpha_{medium\ energetic}$) are stopped in the region behind the depletion layer and the energy that they deposit will be collected slower. Their timing signal is thus larger and in the E/t scatterplot they appear above the recoil line.

High energetic alpha particles ($\alpha_{high\ energetic}$) deposit most of their energy far behind the depletion layer (see the Bragg curve in section 4.1.2) and their timing signals are even slower. The measured energy does not increase anymore, but even decreases for these high energetic alphas. This effect is the *ballistic deficit*, as discussed in section 4.2.2; because the charge collection is too slow, or even incomplete when the charge is deposited beyond the region from which charge is effectively collected by diffusion, not

all deposited charge will contribute to the pulse height that is recorded. This effect causes the alpha curve to fold back in the E/t scatterplot.
Both the undiscriminated and discriminated energy histograms for this sample are shown in figures 4.12 and 4.13, respectively. In the undiscriminated spectrum the alphas form a large 'fold back' peak and obscure the recoil contributions.

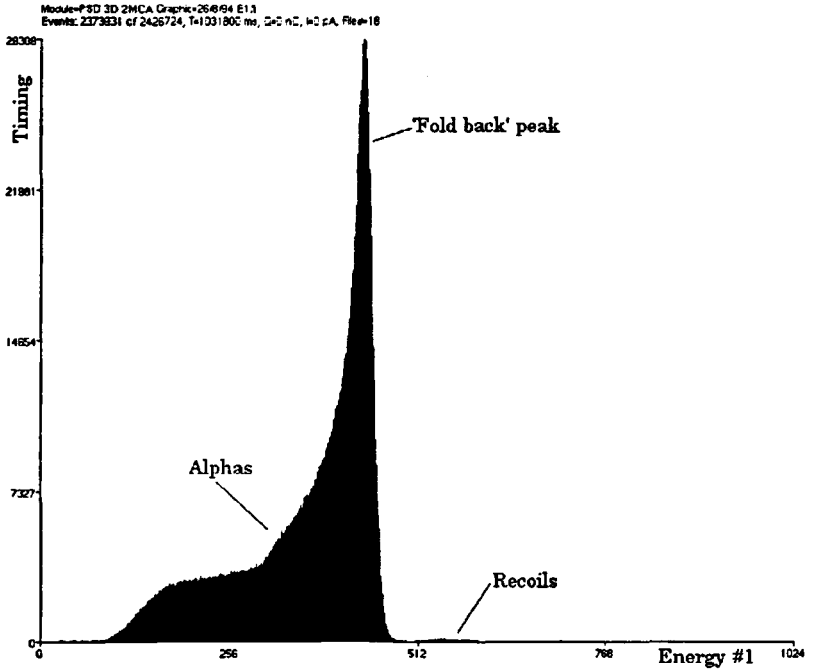

Figure 4.12: The undiscriminated energy histogram projection of figure 4.11

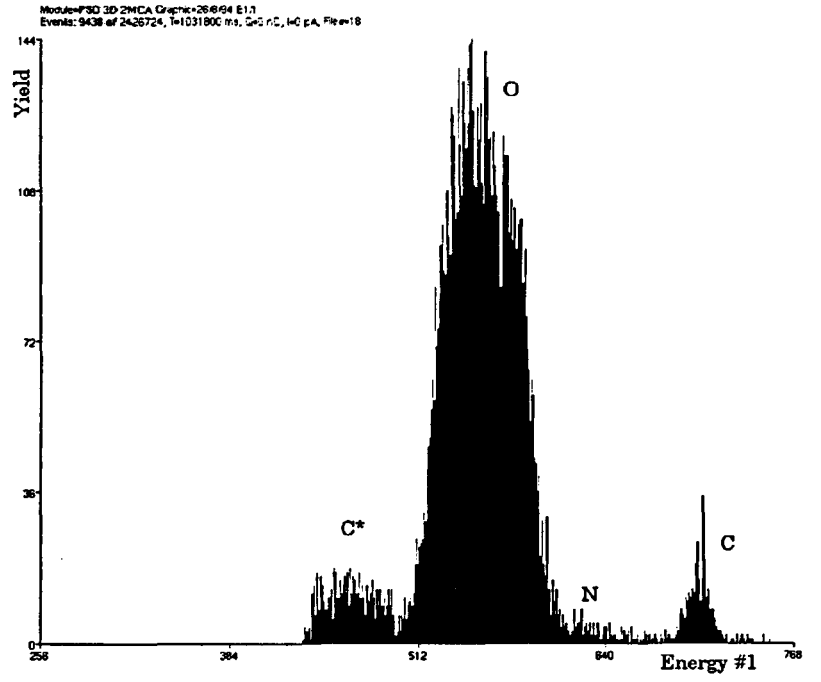

Figure 4.13: The discriminated energy histogram projection of figure 4.11

When the alphas are suppressed by selecting the recoils in the scatterplot, the discriminated, alpha suppressed, energy histogram is obtained in which the recoil contributions are clearly visible. The background from alphas is gone and the pile up background is strongly reduced.

4.5 Optimizing the pulse rise-time discrimination

The measurements that were discussed in the previous section were performed with a pulse rise-time discrimination circuit that was optimized to obtain both a maximum separation between the recoils and the alphas in the E/t scatterplot for a accurate discrimination and a maximum energy resolution. Several combinations of the shaping time for the timing signal, the shaping time for the energy signal and the detector bias voltage were used.

An indication for the level of discrimination is the *alpha suppression energy* which is defined as the maximum energy down to which the alphas can be suppressed reliably. Alphas below this energy have timing signals comparable to recoil timing signals and separation can no longer be based on timing differences. The alpha suppression energy is thus the energy down to which the recoils can be reliably separated from the alphas. A reduction of the alpha suppression energy means an improvement in discrimination.

4.5.1 Different shaping times for the timing parameter

Measurements have been carried out with different values for the shaping time τ_t of the timing parameter (50 ns, 100 ns, 250 ns and 500 ns). The effect of a change in τ_t on the alpha suppression energy is not observed. More measurements are needed to compare the separation and timing resolution for various timing shaping constants.

In further experiments, we used the Ortec 2020 amplifier with the time constant for the bipolar shaping filter equal to 0.25 μs,

4.5.2 Different shaping times for the energy parameter

The shaping time for the energy parameter was also varied to obtain maximum discrimination and optimal energy resolution. Measurement were performed using an energy shaping time τ_E equal to 100 ns, 250 ns, 500 ns, 1000 ns and 2000 ns. The detector bias voltage was 1.0 V, the detection angle φ was 30 degrees and τ_t was 0.25 μs. To show the effect of a change in the energy shaping time, figure 4.14 shows the E/t scatterplot of a 80 nm Si_2O_3N on a Si substrate measured with τ_E=1000 ns. It can be compared to fig 4.11 where τ_E=250 ns. All other conditions were the same.

At higher energy shaping times the alpha curve folds back less, which results in a worse separation between the alpha curve and the recoils. Also, the amount of pile-up increases with a larger shaping time and the recoil contributions are obscured by a background of pile-up.

On the other hand, τ_E should not be taken too small, because if it becomes comparable to the rise time of the recoil energy pulses from the preamplifier, recoil pulse height loss occurs and the energy resolution may become worse.

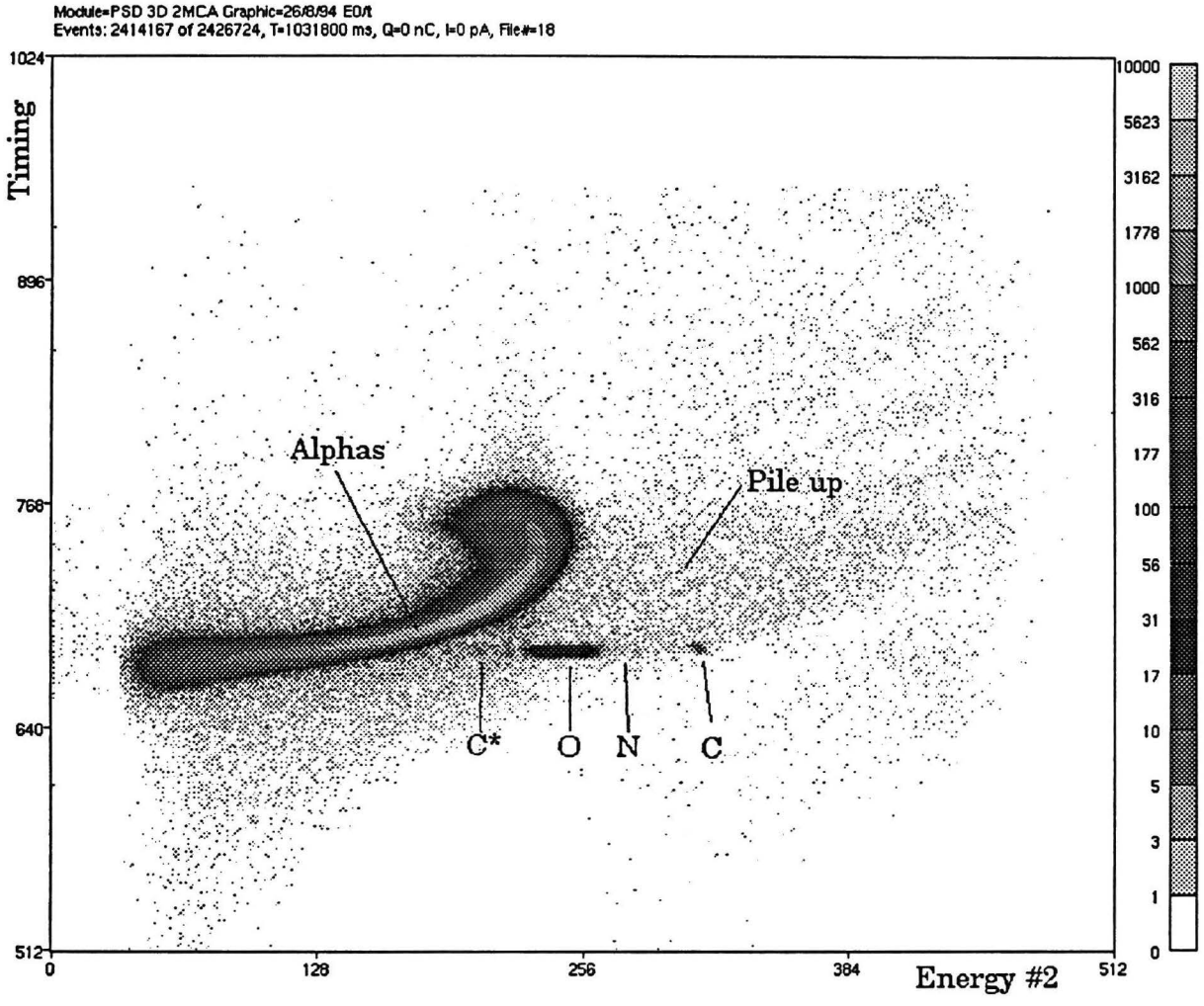

Figure 4.14: The E/t scatterplot of a Si_2O_3N layer on a Si substrate. (φ=30°, ψ=15°, E_0=13.4 MeV, V_b=1.0V, τ_t=0.25 μs, τ_E=1.0 μs)

The total energy resolution for recoils is determined by the beam energy spread, kinematic spread due to angular beam divergence and detection angle spread, the detector energy resolution and finally the energy resolution of the electronics (noise). When the kinematic spread is reduced by using sufficiently narrow slits, the fluctuation in the energy of the recoils that is available for electron/hole production is the detector is more important than all other contributions and the influence of a change in the energy shaping time, thus in the energy resolution of the electronics, on the total energy resolution for recoils will be small.

If the time constant τ_E is set to 0.25 μs, the separation between the recoils and the alpha curve is sufficient to discriminate reliably, while the amount of pile-up is strongly reduced. Because the energy shaping time and the timing shaping time are now taken equal we can use one shaping amplifier for both signals. The unipolar shaping output is used for the energy signal and the bipolar shaping output is used for the timing signal. (MA #1 in figure 4.3). This is a simplification of the electronic pulse rise-time analysis circuit.

Dependance of the energy response on the energy shaping time can **also** be used to discriminate between recoils and alphas. This is done with pulse height discrimination measurements which are discussed in section 4.6.

4.5.3 Different detector bias voltages

The effect of a change in the detector bias voltage was also studied. Measurements were performed with a bias voltage V_b=0.5V, 1.0V, 2.0V, 2.6V and 4.0V. The detection angle was 30 degrees, τ_t=0.25 µs and τ_E=1.0 µs. Figures 4.15 and 4.16 show the E/t scatterplots that correspond to V_b=0.5V and 2.6V, respectively.

Decreasing the bias voltage decreases the depletion layer thickness and the rise time of long range alphas increases. A decrease of the detector bias voltage also causes the alpha curve to fold back further, and the separation between the alphas and the recoils will become better.

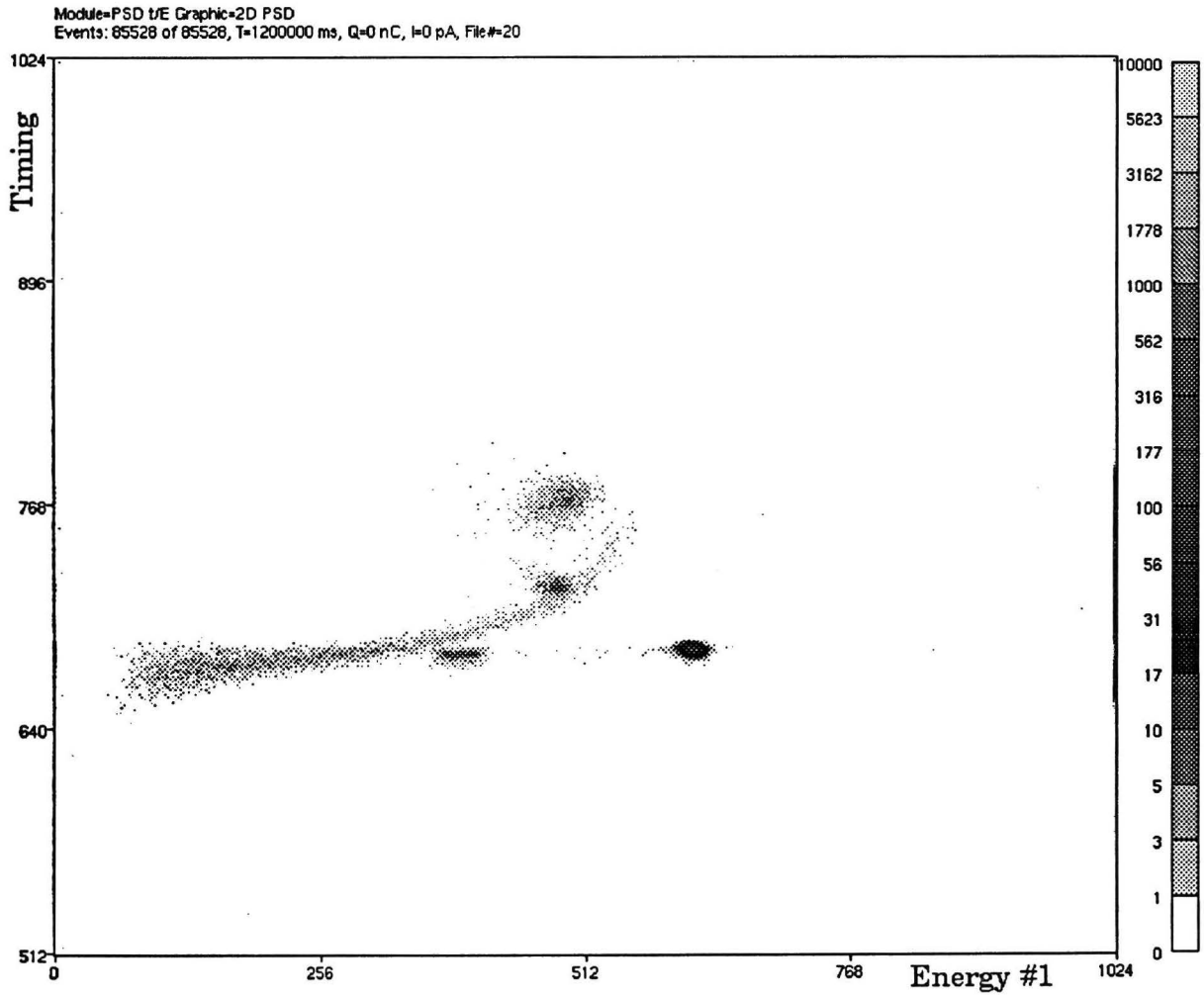

Figure 4.15: The E/t scatterplot of a carbon foil, measured with V_b=0.5V (φ=30°, τ_t=0.25 µs, τ_E=1.0 µs)

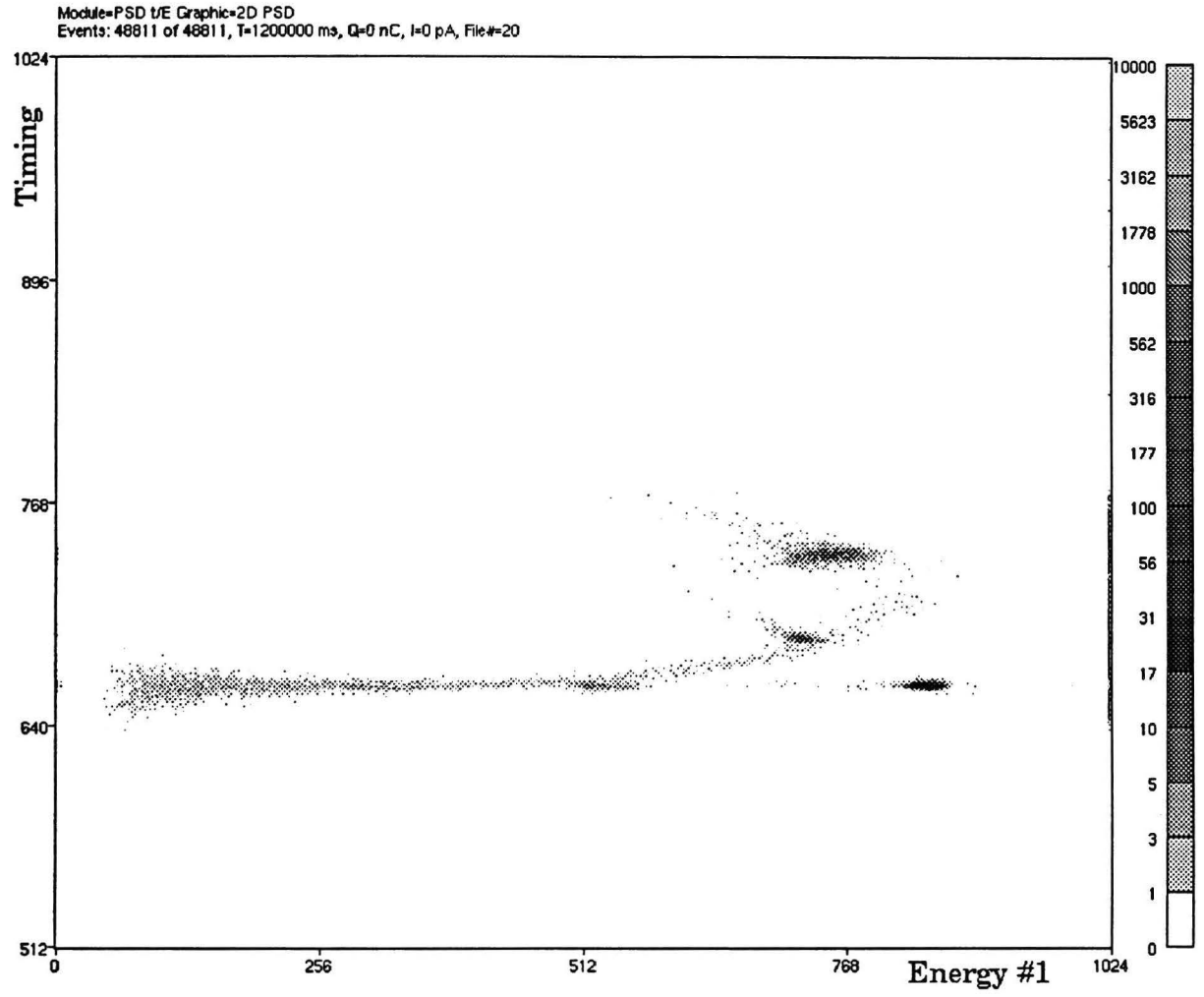

Figure 4.16: The E/t scatterplot of a carbon foil, measured with V_b=2.6V (φ=30°, τ_t=0.25 µs, τ_E=1.0 µs)

At low bias voltages, however, the detector capacity increases strongly [RIJ93] which results in an increase of the rise time of the energy output of the preamplifier and a decrease of the signal-to-noise ratio. The amplification of the preamplifier may even become instable at very high detector capacities (this is discussed in section 4.9.2).
A bias voltage of 1.0V was chosen to keep the detector capacity fairly small, while separation is sufficiently optimized.

4.5.4 An alternative timing circuit

In conventional pulse rise time analysis the start signal for the time-to-amplitude converter is taken from the fast timing circuit while the stop signal comes from the pulse-shape-analyser (PSA) output that gives a fast pulse on the zero crossing of the bipolar energy pulse. This is the so-called B-output of the PSA.
The PSA has a second output, the A-output, that gives a pulse when the down going flank of the bipolar pulse has lost a certain fraction (10%-90%) of its maximum.
Instead of taking the start signal from the fast timing circuit, this A-output pulse from the PSA can be used. Timing differences will be smaller, because only a fraction of the

rise time of the shaped preamplifier pulse is taken into account. But separation between fast and slow pulses remains possible.

Figures 4.17 and 4.18 show the E/t scatterplots of a 390 nm amorphous C layer on a Si substrate, measured with the TAC start signal taken from the fast timing circuit (conventional) and from the PSA A-output (alternative), respectively. These plots show that for both timing methods carbon can be profiled down to a depth of at least 400 nm, because the carbon recoil contribution appears completely separated from the alpha curve. The large amount of pile-up is cause by a high detector count rate. This can be reduced by a reduction of the beam current.

The depth resolution for both methods is the same. In these measurements, the surface depth resolution for oxygen is 40 nm and the depth resolution at the interface is 50 nm, which is slightly larger due to multiple scattering and energy straggling of the recoils.

The main advantage of the alternative rise time analysis circuit is the simplicity; no fast timing circuit is needed and the fast amplifier and the constant fraction discriminator can be done without.

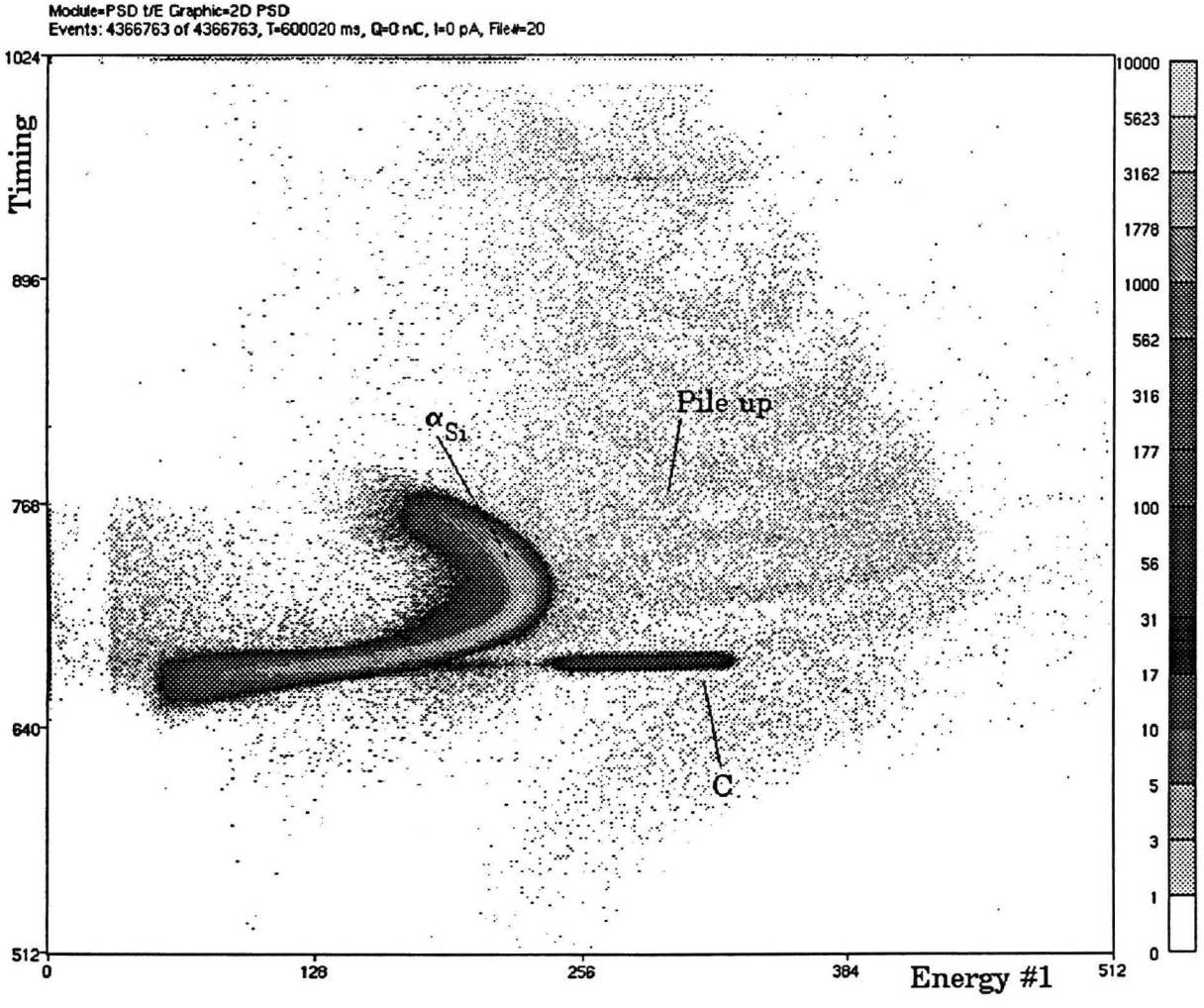

Figure 4.17: The E/t scatterplot of a 390 nm C layer on a Si substrate, measured with the conventional timing circuit. (V_b=1.0V, φ=30°, ψ=15°, $\tau_E=\tau_t$=0.25μs)

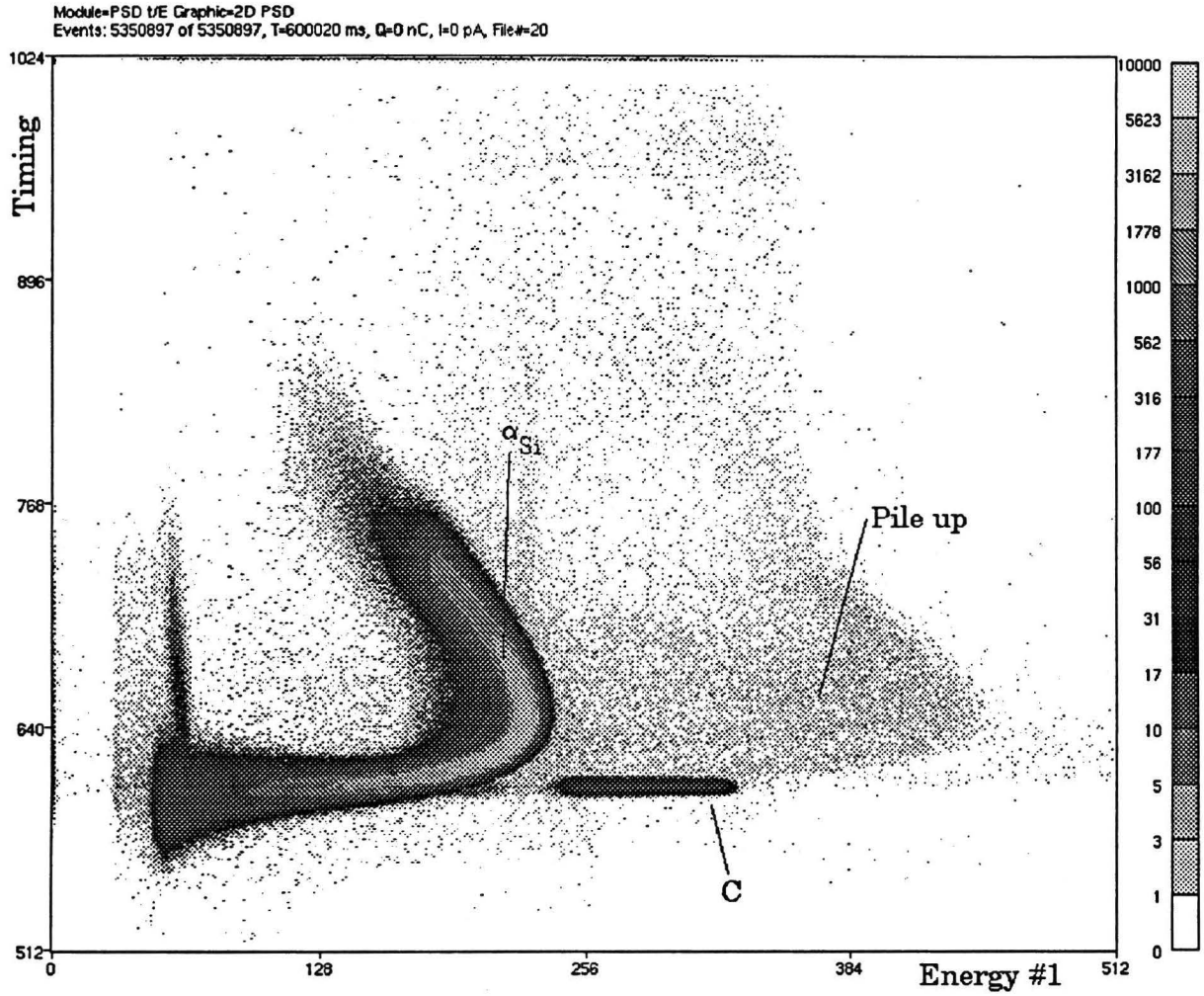

Figure 4.18: The E/t scatterplot of a 390 nm C layer on a Si substrate, measured with the alternative timing circuit. (V_b=1.0V, φ=30°, ψ=15°, τ_E=τ_t=0.25μs)

4.5.5 Different detection angles

A detector with a thin depletion layer only separates recoils from scattered alpha particles, but does not separate the different recoils from one another. The mass/depth ambiguity can be removed by changing the beam energy and/or the detection angle. This will alter the cross section ratio of different detected recoil masses. Cross section resonances can be used to enhance the sensitivity for a nuclide of interest.

For oxygen a broad resonance at 13.4 MeV is found at 30 degrees. For carbon a similar cross section maximum is found at 12.1 MeV, also at 30 degrees. Nitrogen has a cross section maximum at 13.4 MeV at a detection angle of 37 degrees.

Small detection angles are also advantageous: the recoil energy spread is proportional to $dK_{rec}/d\varphi \sim \sin(2\varphi)$ and the depth resolution is thus enhanced for small angles. Furthermore, the recoil mass separation, which is proportional to $dK_{rec}/dm_2 \sim \cos^2(\varphi)$ is optimal at small angles. Finally, very large cross section resonances exist at small recoil detection angles.

Figure 4.19 shows the E/t scatterplot of a measurement of a carbon foil that was bombarded with 13.4 MeV alphas at normal incidence. The detection angle was 15 degrees.

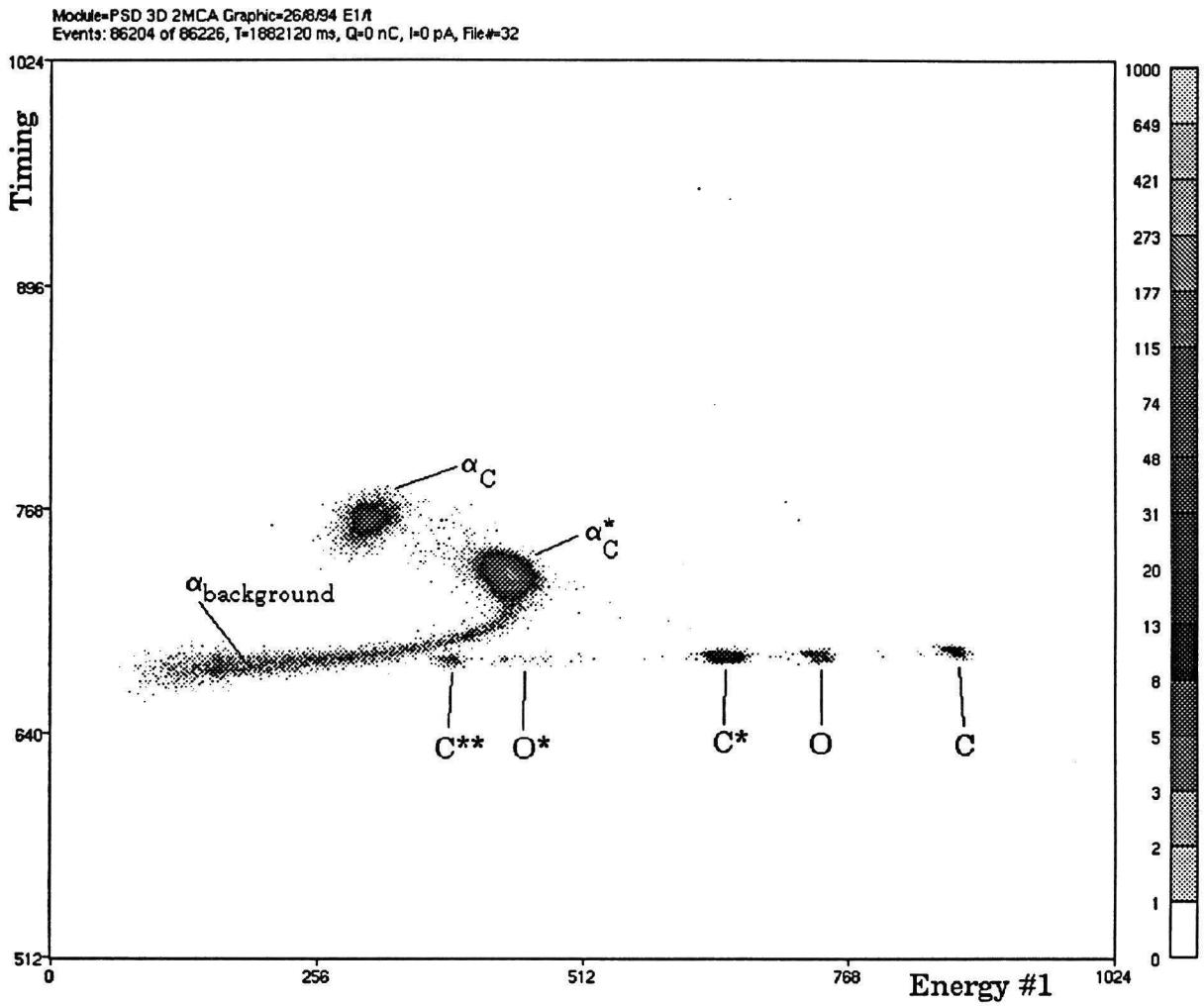

Figure 4.19: The E/t scatterplot of a carbon foil detected at φ=15 degrees. Notice that inelastic oxygen recoils are visible. (V_b=1.0V, E_0=13.4 MeV, φ=15°, ψ=90°, $\tau_E=\tau_t$=0.25μs)

In this measurement both inelastic oxygen and double inelastic carbon recoils appear separated from the alpha tail. A background of alphas of slitscattered alphas that hit the detector directly is also visible (see also section 4.9.1).

Unfortunately, detection at small angles (smaller than 30 degrees) leads to practical problems if the experiment is done with a reflection geometry. A these small detection angles, for which the entrance angle and the exit angle are also small, the lateral spread (due to path length differences, caused by multiple scattering) is increased, extremely flat samples are required and the beam spot size on the sample has to be minimized. First of all, the target mounting device should be redesigned, because the current design is not fit for small detection angles in reflection geometry.

4.6 Pulse height discrimination measurements

As indicated in section 4.5.2 discrimination can also be based on the dependance of the energy response to the energy shaping time. This is demonstrated by a measurement on a carbon foil and a measurement on a thick sample.

4.6.1 A thin carbon foil

Instead of drawing an E/t scatterplot of the timing parameter and one of the energy parameters, the two energy parameters with different shaping times τ_{E1} and τ_{E2} can also be plotted versus each other in a scatterplot. Such an E1/E2 scatterplot is shown in figure 4.20. It was created from the same list mode data that was used to create the E/t scatterplot in figure 4.5.

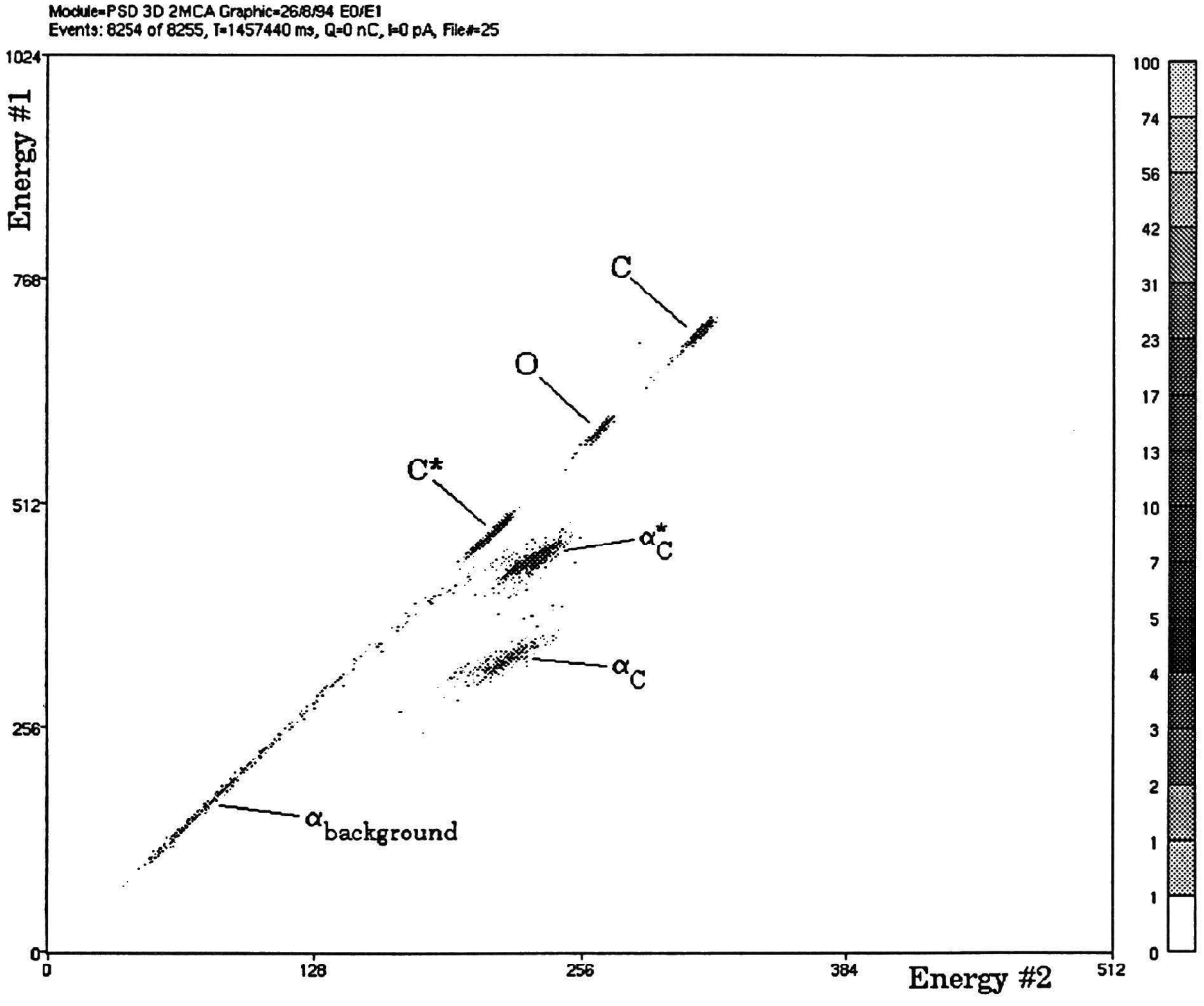

Figure 4.20: The E1/E2 scatterplot of a carbon foil. τ_{E1}=0.25μs and τ_{E2}=1.0μs (φ=30°, ψ=90°, V_b=1.0V)

The energy parameter $E_{\tau 1}$ (shaping time τ_{E1} equal to 0.25 μs) is plotted along the vertical axis and the energy parameter $E_{\tau 2}$ (shaping time τ_{E2} equal to 1.0 μs) is plotted along the horizontal axis. Units are still in channels.

Particles that are stopped in the depletion layer, like recoils, have a similar response to shaping with different times constants, that are all larger than the rise time of the energy pulses, and in the E1/E2 scatterplot they form a straight line. In figure 4.20 the contributions from the elastic carbon recoils (C, 7.5 MeV), the inelastic carbon recoils (C*, 5.0 MeV) and the elastic oxygen recoils (O, 6.4 MeV) indeed form a straight line.
Alphas scattered on carbon are stopped beyond the depletion layer and because their detector output pulse is slow, they suffer some pulse height loss. The magnitude of this ballistic deficit is a function of the shaping time. Pulse height loss is larger for shorter shaping times. This results in a deviation from the straight recoil line. The alphas that scattered elastically on carbon (indicated by α_C) and the alphas that scattered inelastically on carbon (α_C^*) do indeed appear separated from the recoil line.
Again, a contour can be drawn than encloses the contributions of the recoils. Axis projections can be created of the contents of the contour and a alpha suppressed energy histogram, like in figure 4.8, is obtained for each energy parameter.

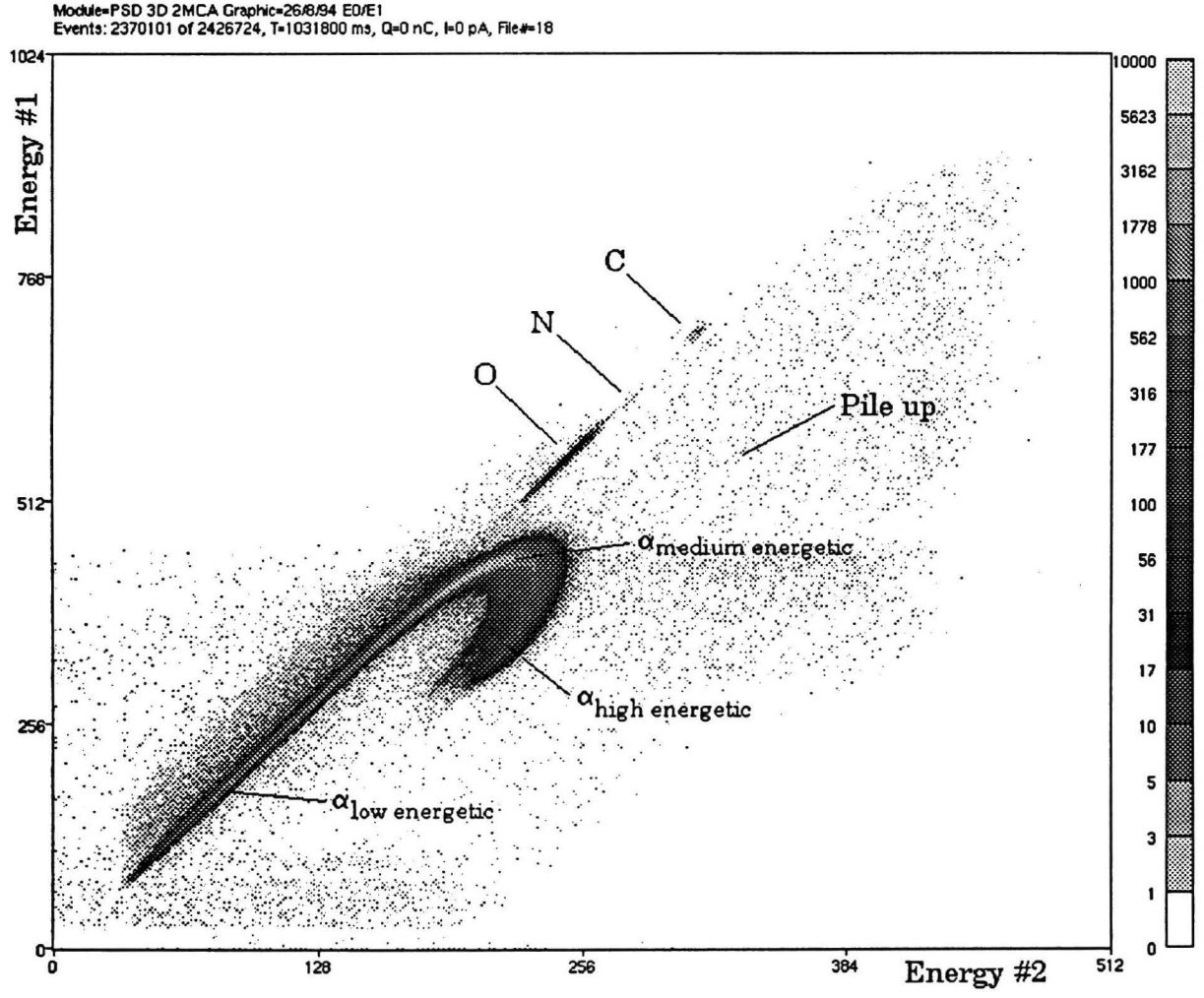

Figure 4.21: The E1/E2 scatterplot of a Si_2O_3N layer on a Si substrate. (φ=30°, ψ=15°, E_0=13.4 MeV, V_b=1.0V, τ_{E1}=0.25μs, τ_{E2}=1.0μs)

4.6.2 A Si_2O_3N layer on a Si substrate

The contributions of alphas to an E1/E2 scatterplot can be seen better for a thick sample. Figure 4.21 shows the E1/E2 scatterplot that corresponds to the E/t scatterplot in figure 4.11.
The elastic carbon recoils (C), the oxygen recoils (O) and a few nitrogen recoils (N) again fall on the straight recoil line.
The curve of scattered alphas, mostly on Si, folds back for high energetic alphas. As explained in the previous section this is due to the pulse height loss that is dependent on the shaping time constant. A pile-up background caused by a high detector count rate is also visible.

4.7 Optimizing the pulse height discrimination

4.7.1 Variation of the energy shaping times

Different combinations of energy shaping times were used to see whether the separation between the recoils and the alpha tail could be improved. It was found that an increase in the ratio of the energy shaping times τ_{E1} and τ_{E2} improved the separation between the alpha curve and the recoil contributions. The shaping constants should not be too large, to reduce the amount of pile-up. If the time constants are taken too small, pulse height loss for the recoil energy pulses occurs and the recoil contributions will no longer form a straight line.
The combination of τ_{E1}=0.25μs and τ_{E2}=1.0 μs (and V_b=1.0V) was found to be useful for effective pulse height discrimination and a combination with pulse rise-time analysis (with τ_t=τ_E=0.25 μs) can then easily be accomplished.

4.8 Other improvements

4.8.1 Alpha suppression with the CFD threshold level

The constant fraction discriminator (CFD) has a threshold level that is usually used to suppress noise pulses from the fast timing output of the preamplifier; if the target is not bombarded with projectiles, any fast timing signals must be due to noise and these are suppressed by increasing the threshold level just over the noise level.
The threshold level can be increased further to suppress the fast timing pulses from both low energetic particles and long range particles. This suppression of alphas at both ends of the alpha curve is shown in figure 4.22. Compare figure 4.22 with figure 4.11, for which the CFD level was much smaller.
If the CFD threshold level is increased too much, recoil events are also suppressed. Suppressing all alphas without suppressing a recoil event seems impossible and increasing the CFD level can not be used to reliably suppress **all** alpha events.
It can, however, be used to suppress a part of the alphas and thus reduce the amount of events that has to be processed by the multi-parameter data-acquisition system. The amount of data is reduced and list mode data can be processed quicker.

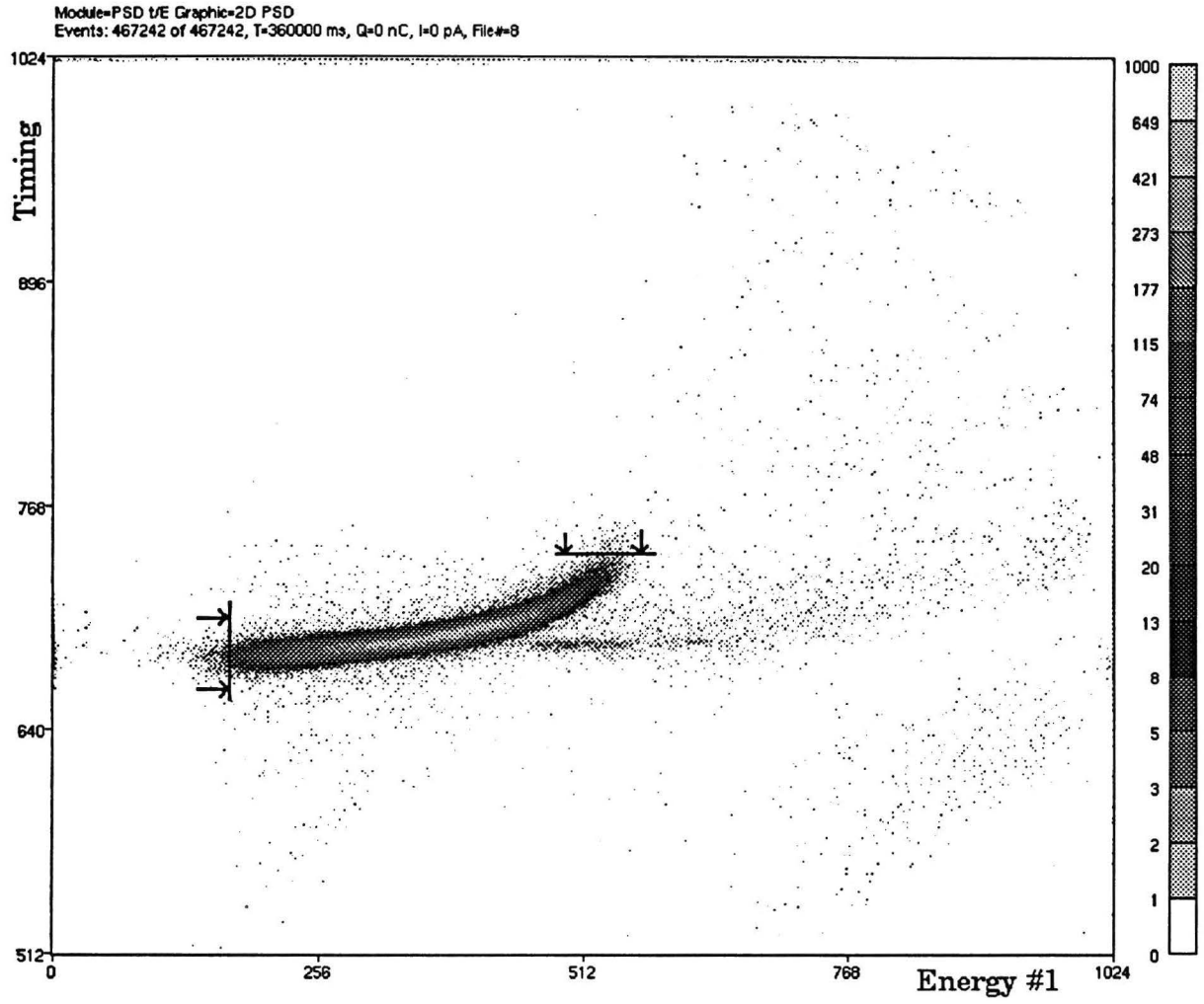

Figure 4.22: Suppression of the alpha curve at both ends by increasing the CFD level (Si_2O_3N/Si, φ=37°, ψ=15°, E_0=13.4 MeV, V_b=1.0V, τ_E=1.0μs, τ_t=0.25μs, CFD level 80)

A suggestion for future experiments is to include the fast timing output of the preamplifier as an extra experiment parameter and to use the multi-parameter system instead of the CFD threshold level to discriminate on this parameter.

4.8.2 Discrimination based on more than two parameters

With the setup that is described in section 4.3 three coincident parameters are recorded: one timing parameter and two energy parameters. Up to now only two out of these parameters are used to discriminate between the recoils and the alphas. A discrimination method that is based on more than two parameters might be more effective. This is indicated by the following example.

When the three scatterplots are drawn (the E1/t, the E2/t and the E1/E2 scatterplot) and when in all three plots a contour is drawn that encloses the recoil contributions, a small difference in the number of selected recoil events for each scatterplot is obtained. This difference is among others caused by pile-up which is not the same for all three parameters. It should be possible to reduce the pile up background by a discrimination method that is based on more than one parameter. This might improve sensitivity.

It is not yet possible to select a contour in a scatterplot of two parameters and then create the scatterplots of the other two combinations of parameters with only those events that lie within the contour in the first scatterplot. An option like this would allow flexible analysis based on more than two parameters.
It is however possible to set a condition on a parameter. It is thus, for example, possible to set an interval for the timing parameter and then draw the scatterplot for the energy parameters with only those events for which lies within the interval.

4.9 Encountered problems and effects

4.9.1 Slitscattering background

During measurements on a carbon foil, it was noticed that the alpha background curve bypassed the peak from alphas that scattered inelastically on carbon. The background alphas had a larger pulse height for the same timing signal This is illustrated in figure 4.23.

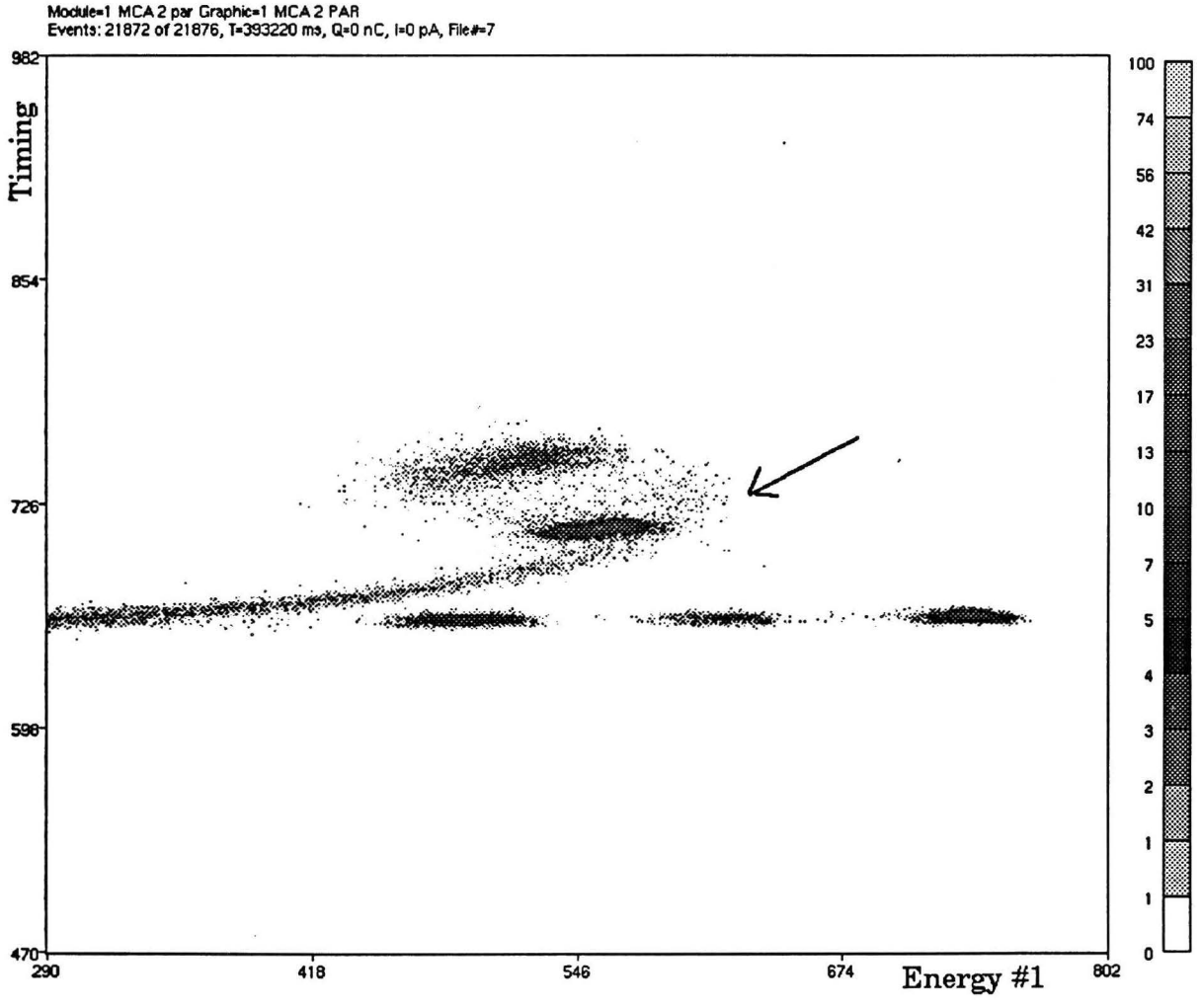

Figure 4.23: A E / t scatterplot of a carbon foil. The alpha background curve bypasses the peak from the alphas that scattered inelastically on carbon.
(The slit in front of the detector was removed for this measurement)

The hypothesis was that these alphas were not scattered on the target but that they entered the detector directly from the beam guidance system, where they were scattered on the diaphragm placed just in front of the scattering chamber. Because they crossed the depletion layer obliquely, they deposited more of their energy within the depletion layer. The rest of the energy was deposited closer to the depletion layer and thus the amount of charge collected by diffusion was also larger. This explains the shift of the alpha background curve to a larger pulse height.
The hypothesis is confirmed by a measurement without a target where only the shifted alpha background curve remained visible, and by a measurement with a different diaphragm which caused a change in the intensity of the background. A shield has been installed in the target chamber to block all alphas that are scattered on the diaphragm and this removes the background almost completely.

4.9.2 Variation in the channel to energy calibration

A channel to energy calibration can be made for recoils, based on the location of recoil peaks with a known energy. The energy calibration should remain fixed during an experiment session, if nothing is changed but the sample.
It was observed, however, that over a number of consecutive measurements (over a time interval of several hours) the channel to energy calibration did **not** remain fixed. The location of the carbon peak, for example, shifted to a lower channel. This shift was observed for both energy parameters and was not present in the spectrum of the monitor energy parameter. The effect is thus limited to the PSD detector and the corresponding preamplifier. A variation of the beam energy can be excluded.
The preamplifier gain is stable as long as the detector capacity C_d is much smaller than the input impedance $A \cdot C_f$ of the preamplifier (where A is the amplification factor of the amplifier and C_f is the feedback capacity) [RIJ93]. However, at small bias voltages C_d becomes large and the gain of the preamplifier might become instable and vary with A and C_d
The detector capacity C_d may change through a change of the depletion layer thickness of the detector. A variation in the depletion layer thickness may have two causes.
First, a change of the applied bias voltage changes the depletion layer thickness. A slow response of the depletion layer thickness to a change in the applied bias voltage might explain the observed variation. However, this response time, which depends on the time constant of the bias voltage low pass filter in the preamplifier, is at most a few seconds.
A second cause might be that at high count rates the detector leakage current increases and the effective bias voltage decreases. This also decreases the depletion layer thickness and increases the detector capacity.
It is more likely that the observed pulse height changes are not caused by a change of the depletion layer thickness, but by a change of the preamplifier gain which is instable because the detector capacity C_d is high.
The amplification factor A can change if high count rates cause temperature changes or a change in the amplifier current to compensate the DC-level when pulses are superimposed.

A number of experiments was done to test the hypotheses that were give above. A measurement with an AmCu alpha source, that was started immediately after the detector bias voltage had been changed from 100V down to 1.0V, did not show a shift of the peaks. The count rate during this experiment was low. It shows that the response of the depletion layer thickness to a change in applied bias voltage is a matter of seconds.
Another experiment was performed to see if the count rate indeed influences the pulse height gain. An AmCu alpha source was placed in front of the PSD detector with 1.0V bias. The detector was connected to the detector input of the preamplifier and two pulse generators were connected to the test input of the preamplifier, each with an output pulse similar to a detector output pulse. One of the pulse generators produced a 50Hz signal with a constant amplitude, the other produced pulses with a variable frequency.
An increase of the count rate of the variable pulser caused a decrease of the measured pulse height of both the alpha source pulses and the fixed gain test pulse. This shows that the gain of the preamplifier is indeed instable at high count rates. The same effect was observed for another preamplifier of the same type.
As a temporary solution, the beam current was reduced during consecutive experiments to decrease the count rate. Thus, the gain shift was avoided **and** the pile-up background was reduced. For a final solution a new preamplifier is needed that can handle high count rates in combination with a high capacity connected to its input.

4.9.3 Broadening and splitting of the alpha peaks and curve

For a number of experiments a splitting of the high energetic alpha peaks and curve was observed; in addition to each expected peak, two extra peaks appeared with a lower pulse height for each energy parameter. The intensity was also smaller. This is illustrated by figure 4.24. For an E/t scatterplot the extra peaks appear with a faster timing signal.

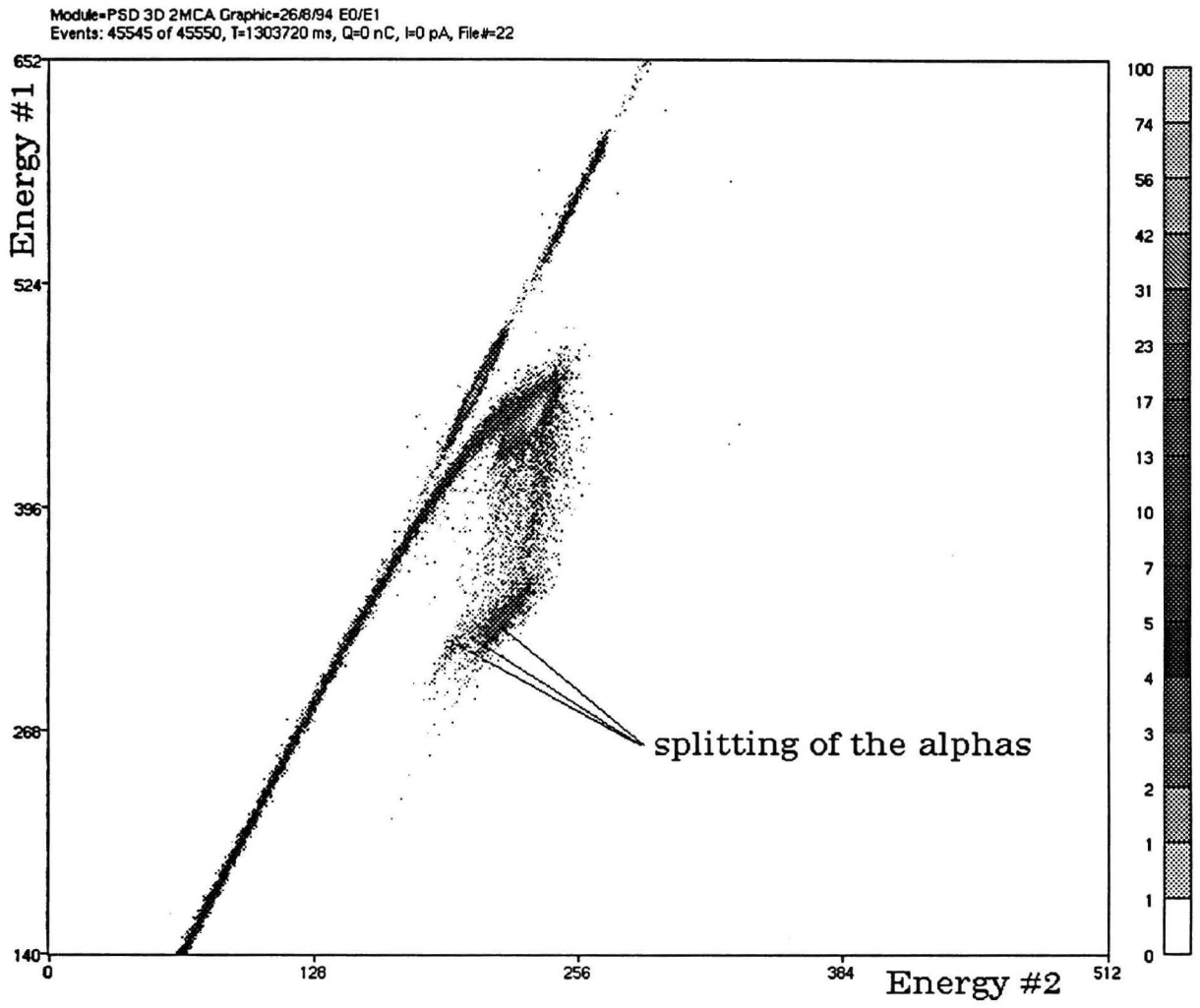

Figure 4.24: The splitting up of the alpha peaks and curve (carbon foil, V_b=1.0V, φ=30°, ψ=15°, E_0=13.4 MeV)

At first, the splitting of the high energetic alphas was considered to be due to cable reflections in the electronic circuit. A test series in which the cable connections were varied (adding 50Ω terminators and replacing tee-splitters) did not show any improvement.

Multiple beam energies due to an extraction problem are most unlikely: the monitor energy parameter did not exhibit the observed effect.

However, the splitting-effect was not observed with a different detector (and the same preamplifier). Figure 4.25 shows the measurement that was done with this other PSD detector.

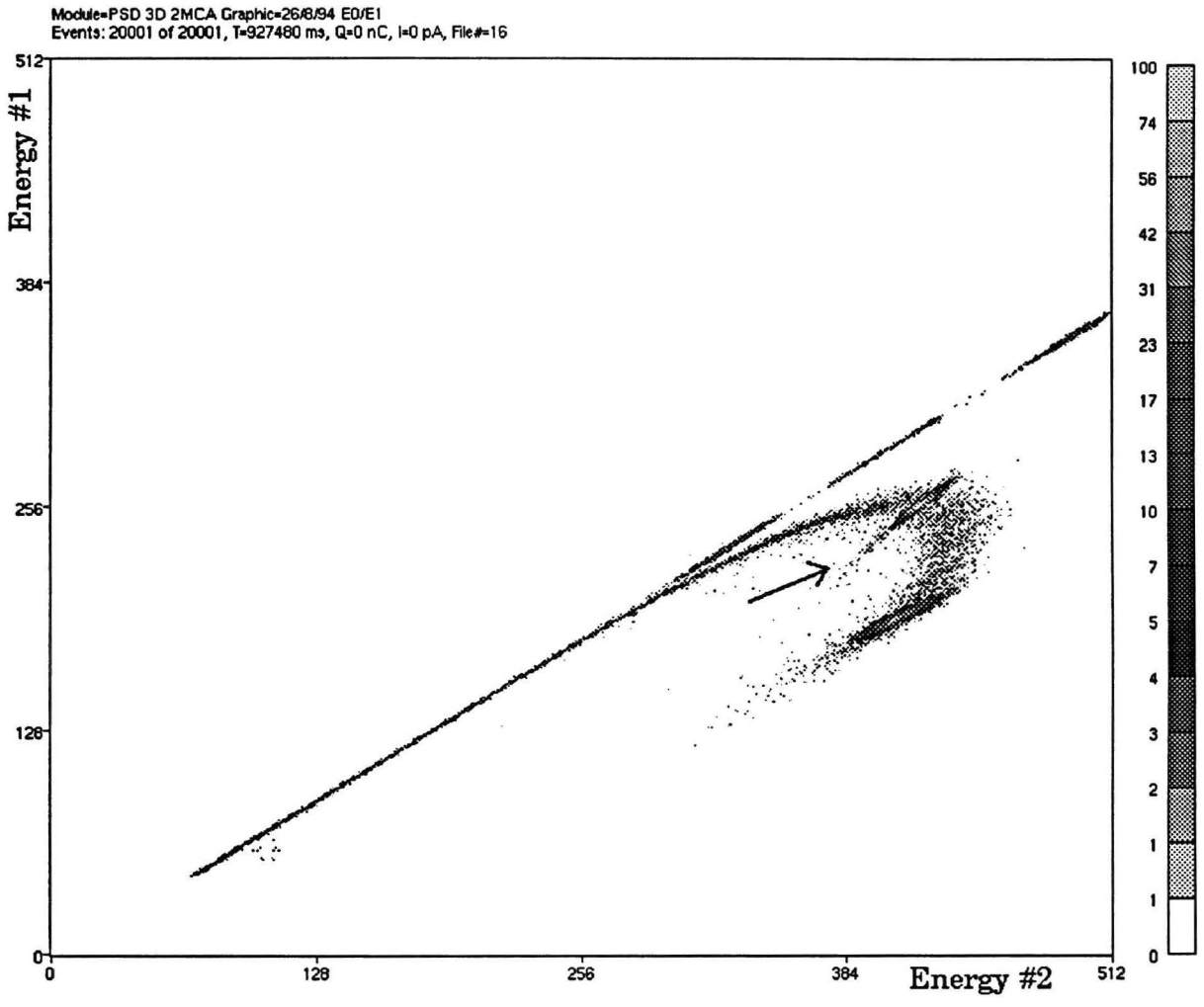

Figure 4.25: A different detector does not show the splitting of the alpha curve to lower pulse heights. Broadening of the alpha peaks through straggling and multiple scattering appears unobscured. (carbon foil, V_b=1.0V, φ=30°, ψ=15°, E_0=13.4 MeV)

With the new detector the splitting of the alpha peaks to lower energy and faster timing was gone, suggesting that it was caused by detector defects, like radiation damage. These damages are located in the region beyond the depletion layer, because the splitting is only seen for high energetic alphas, and the pulse height loss is correlated to the shaping time.

For this detector the broadening of the alpha peaks and their tail to lower energies and **slower** timing is not obscured by the splitting of the alpha curve. The broadening may possibly be explained by energy straggling and multiple scattering of the alpha particles in the detector and small variations of the depletion layer thickness over the detector area. The tail is probably caused by 'trapping' effects caused by parts of the detector that are damaged by radiation and from which charge collection is slow and inefficient.

Chapter 5
Conclusions and recommendations

5.1 The multi-parameter data-acquisition software

Software has been developed for the PhyDAS data-acquisition system to perform multi-parameter experiments in both list mode and histogram mode. It can be used for microbeam experiments and for ion scattering experiments. If routines are added to control the channeling set-up, it can also be used for channeling experiments.

A specification and a design have been made for a new version of Columbus that can both process and monitor data collected by the PhyDAS multi-parameter data-acquisition system. The user interface and the data base were completely rewritten and data of multi-parameter list mode experiments can be processed and monitored. The monitoring of histogram mode experiments and an equation interpreter to process parameters into derived parameters still have to be added. The tool to make axis projections of the contents of a contour has to be extended, so projections can be made onto any user-defined axis. This might also be done by a transformation and rotation of the parameters, using the equation interpreter.

5.2 Multi-parameter Pulse Shape Discrimination

Pulse Shape Discrimination experiments have been performed with the new multi-parameter data-acquisition system. Because experiment data is monitored properly and because relations between parameters are clearly visible, understanding of the Pulse Shape Discrimination technique has improved.

Separation between the alpha and the recoil contributions, which is essential in ERDA experiments has been improved and discrimination is now more accurate. This results in an improved sensitivity for recoils.

An alternative timing circuit has been developed. This timing circuit only uses the energy output of the preamplifier and the fast timing circuit can be done without. Separation between the alphas and the recoil contributions is slightly worse, but depth resolution is not influenced.

In addition to discrimination based on pulse rise-time analysis, a second discrimination technique was found to be useful. This discrimination is based on the analysis of the pulse-height response of the energy pulses to shaping with different shaping constants and is as effective as the rise-time discrimination. The electronics is simplified even more, because the timing circuit can be done without completely.

Discrimination based on a combination of both pulse rise-time analysis and pulse height analysis might improve separation and thus sensitivity even more.

Finally, all data that was collected during the experiments still awaits quantitative analysis, which is now possible.

References

[ATZ92] B. Atzema
Data acquisition for microbeam analysis
Final report of the postgraduate program Software Technology, EUT, 1992

[BEU94] L. van Beurden
Channeling met hoog energetische ionenbundels
Master of Science thesis (VDF/NK 94-22), EUT, 1994

[DIJ92] P. W. L. van Dijk
Analysis of light elements in thin films using high energy ion scattering techniques
Master of Science thesis (VDF/NK 92-02), EUT, 1992

[FEL86] L. C. Feldman, J. W. Mayer
Fundamentals of surface and thin film analysis
Elsevier Science Publishing, Amsterdam, 1986

[GER94] P. J. German
Een database voor experiment-gegevens. Database in Oracle met een interface voor de taal C
Afstudeerverslag Elektrotechniek TCK Hogeschool Eindhoven, 1994

[IJZ93] L. J. van IJzendoorn, H. A. Rijken, S. S. Klein, M. J. A. de Voigt,
Applied Surface Science, 70/71 (1993) 58

[JAE94] L. Jaegers
Elastic recoil detection analysis with He ions. Simulations and applications to materials analysis
Master of Science thesis (VDF/NK 94-15), EUT, 1994

[JON93] J. Jonkers
Multi-Parameter Ion Scattering Techniques
Master of Science thesis (VDF/NK 93-24), EUT, 1993

[KNO79] G. F. Knoll
Radiation detection and measurement
Wiley & Sons, New York, 1979

[RIJ93] H. A. Rijken
Detection methods for depth profiling of light elements using high energy alpha particles
Ph.D. thesis, EUT, 1993

[SIM93] D. P. L. Simons
Ontwikkeling van de software voor het uitvoeren van histogrammode-

experimenten met de microbundelopstelling
Internal report (VDF/NK 93-30), EUT, 1993

[STR93] G. J. Strijkers
On line grafische verwerking van multi-parameter experimenten op PhyDAS
Internal report (VDF/NK 93-19), EUT, 1993

[ZWI93] C. van Zwijnsvoorde
Software ontwerp en implementatie voor metingen aan spoorelementen in de scannende protonen microbundelopstelling.
Master of Science thesis (VDF/NK 93-03), EUT, 1993

Appendix A
Changes to the CEDAS software levels

A.1 Changes to the routines level

Routines have been added to the routines level *lvl3* for defining multi-parameter set-ups for PhyDAS. The set-up is stored via a module definition containing the relations between parameters, modules, MCA's and list mode memories.

A.1.1 Variables and constants

- *max_nr_of_modules* = 3
  The maximum number of modules that can be uses in the set-up. Restricted by the number of MCA's in the PhyDAS system.
- *max_nr_of_mems* = 6
  The number of list mode memories that is available.
- *max_nr_of_mcas* = 3
  The number of MCA interfaces that is available.
- *max_mod_mca_size* = *max_nr_of_mcas*
  The maximum number of MCA's in a module
- *max_mod_mem_size* = *max_nr_of_mems*
  The maximum number of memories in a module
- *max_mod_par_size* = 2**max_nr_of_mems*
  The maximum number of parameters in a module
- *nr_of_modules*: nat1
  The number of modules in the current module definition.
- *mod_mca_size*, *mod_mem_size*, *mod_par_size*: ARRAY 1..*max_nr_of_modules* OF nat1
  The number of resp. MCA's, memories and parameters for every module.
- *mem_index*, *mem_mode*: ARRAY 1..*max_nr_of_modules*, 1..*max_mod_mem_size* OF nat1
  The index and memorie mode (16 or 32) of every memory in every module
- *mca_index*, *mca_mode*: ARRAY 1..*max_nr_of_modules*, 1..*max_mod_mca_size* OF nat1
  The index and MCA mode (single or dual) of every MCA in every module.

A.1.2 Routines

- **save_module_info**
  Saves the current module definition in a file called `ubd:modXXX.ubd` where `XXX` is the module definitions number, as given by the user. The file contains the following data:
  - the number of modules in the module definition.
  - the number of memories, mcas and parameters for every module.
  - the indices and memory-modes for all memories.
  - the indices and MCA-modes for all MCA's.
- **load_module_info**
  Reads a module definition from a previously saved file (see save_module_info). The number of the module definitions file is taken from user input.
- **show_module_info**
  Displays an overview of the current module definition.
- **change_module_info**
  Allows the user to change the current module definition. User input is required for:
  - the number of modules..
  - the number of list mode memories for every module.
  - the name (or index) for every list mode memory.
  - the number of parameters for every memory (one or two).
  - For every memory that is connected to a MCA, the name (or index) of that MCA.

From this data the module definition is created and displayed.

- **start_mcas**
  This procedure is used to start all MCA's in all defined modules at the same time. It uses the procedure **strt(*mca_index[i, j]*)** to set the start bit of the MCA's and then it enables all MCA's at once via the relais-interface.
- **stop_mcas**
  This procedure stops all MCA's for all defined modules at the same time. First it disables all MCA's via the relais-interface, and then it uses the procedure **stp(*mca_index[i,j]*)** to stop all of them.

A.2 Changes to the controlling level for list mode experiments

The controlling level for list mode experiments has been expanded into a full multi-parameter list mode experiment controlling level. In contrast to previous versions it is now possible to measure coincident parameters and use both ADC's of a MCA-board during list mode experiments.

A.2.1 Variables and constants

- *header_length* = 2048
  The length of the header in 16-bit words of the list mode files.
- *dirnam*: ARRAY 3 OF char
  The directory on the hard disk in which the list mode files are stored by the data server.
- *start_file_nr*: nat2
  The file number of the first file of the list mode file series.
- *version_char*: char
  Used in creating the file names: indicates the version of the list mode series.
- *nr_of_samples*: nat2
  The number of samples taken during the list mode experiment. Corresponds to the number of files in the list mode file series.
- *mod_char*: ARRAY 1..*max_nr_of_modules* OF char
  The module identification character for every module. Used for creation of the file names.
- *tl_file_name_format*: ARRAY 16 OF char
  String containing the list mode file name format.
- *nr_of_events*: ARRAY 1..*max_nr_of_modules*, 1..*max_mod_mem_size* OF integer
  Array for storing the number of events in every list mode memory for every module. Since all parameters in a module are coincident, the number of events for every memory in one module should be equal.
- *mem_data*: LARRAY 1024**max_nr_of_mems* OF nat2
  Array for temporary storage of the contents of the list mode memories when reading them out and saving the contents via the data server.

A.2.2 Routines

- **splm**
  Show Parameters List Mode Allows the user to change the parameters of the list mode experiment interactively. Parameters that can be changed are e.g. the sample time, the number of samples, the module identification and version characters needed for the file names.
- **stlm**
  Start List Mode. Performs a list mode experiment based on the parameters set by the user. The following routines are used by this routine.
- **initialize**
  Resets the list mode memories to the required memory modes and also resets to MCA's to the required MCA modes. This is done according to the module definitions from *lvl3*.
- **read_all_mems**

Reads the contents ofll list mode memories in all modules into the variable *mem_data[ ]*. The number of events for every memory is stored into the variable *nr_of_events[ ]*.

- **send_mca_data_to_vax(*last*: nat1)**
  Saves the list mode data from the variable *mem_data[ ]* via the data server into files on the hard disk of the Alpha. The exact file format is discussed in appendix B.

A.3 Changes to the controlling level for histogram mode experiments

This level has also been adapted to the new multi-parameter requirements. The level now also uses the module definitions from *lvl3*, so list mode experiments and histogram mode experiments can be done without having to save the input of the parameters into the PhyDAS system.

A.3.1 Variables and constants

- *max_nr_of_hist_mems* = 2 * *max_nr_of_mcas*
  The number of histogram memories available in the system.
- *nr_of_hist_mems*: integer
  The number of histogram memories in use.
- *dir_nam*: ARRAY 3 OF char
  String containg the directory name.
- *tl_file_name_format*: ARRAY 16 OF char
  This variable contains the file name format.
- *mod_char*: ARRAY 1..*max_nr_of_modules* OF char
  The array containing the module identification characters for every module.
- *version_char*: char
  Contains the version identification character for use in the file name.
- *start_file_nr*: integer
  The file number of the first file in the file series.
- *nr_of_samples*: integer
  The number of samples taken during the experiment
- *mca_histogram*: ARRAY 1..*max_nr_of_hist_mems*, 0..4095 OF integer
  The array used for intermediate storage of the contents of the histogram mode memories before saving this data to files.
- *nr_of_events_mca*: ARRAY 1..*max_nr_of_hist_mems* OF integer.
  Array for storage of the number of events in every histogram memory.

A.3.2 Routines

- **sphm**
  Show parameters histogram mode. Displays a menu via which the user can change the experiment parameters, like sample times etc.
- **stsp**
  Start spectrum mode experiment. Starts the histogram mode experiment that is done according to the parameters as set by the user.
- **init_spec**
  Resets all MCA that are to be used in the experiment and initialises *nr_of_hist_mems* and *mca_histogram[ ]*.
- **read_hist_mems**
  Copies the contents of the MCA histogram memories into the array *mca_histogram[ ]*. The contents of the histogram memories are not cleared.
- **send_mca_histogram_tovax(*last*: nat1)**
  Writes the data from *mca_histogram[ ]* into files on hard disk. File names and file format is according to the new standard as discussed in appendix C.

Appendix B
The list mode file format

- Binary file of 16-bit words
- File name: CPV_XXXX.DAT with
  C the module identification character
  P the memory in the module that the data comes from (starting at 0)
  V the version identification character
  XXXX the file number (starting at 0)
  DAT the extension
  Example: A1B_0020.DAT: file with the data of the 2nd memory from the module with id. character A. This is version B of this file and the filenumber is 20.
- The file contains a header followed by a data area. The header length is flexible (currently set to 2048 16-bit words) and the length of the data area depends on the number of events in the file.
- The header format is as follows:

  header[0] = length of the header in 16-bit words (currently 2048)
  header[1] = low word of the number of events in this file
  header[2] = memory mode: 16 if only only parameter and 32 if two parameters are in this file.
  header[3] = last file indicator: 2 if not the last file in this series, 3 if it is the last file in this series
  header[4] = high word of the charge
  header[5] = low word of the charge
  header[6] = low word of the real sample time in units of 10ms
  header[7] = dead time in ms
  header[8] = high word of the number of events in this file
  header[9] = high word of the real sample time in units of 10ms

  The rest of the header is filled with zeros.
- The data format is as follows:
  If the memory mode is 16 and the list mode data of only one parameter is stored in this file then the data are consists of a sequence of 16-bit words corresponding to the values of this parameter for every event. If the memory mode is 32 and the data of two parameters is stored in this file, then the file will consist of a sequence of 2*16-bit words, one for each parameter for every event.

Appendix C
The histogram mode file format

- ASCII text files, case INsensitive
- File name: CPV_XXXX.SPE with
  C the module identification character
  P the parameter number within the module (starting at 0)
  V the version identification character
  XXXX the file number (starting at 0)
  SPE the extension
  Example: A1B_0020.SPE: file with the histogram of the 2nd parameter from the module with id. character A. This is version B of this file and the filenumber is 20.
- The file consist of several main items starting with the main item name ($...) followed by several lines of data (subitems).
- The first main item should be $SPEC_ID: and the last main item must be $END:. The order of the other main items is unimportant.
- The order of the subitems is fixed. New subitems should be added at the end.
- Empty lines between the main items are allowed.
- We will now discuss the available items:

$SPEC_ID:		
{text}		file description (for user only). Maximum of 80 chars.
$EXP_INFO:		
T {time}	INT	real sample time in 1ms
DT {dead time}	INT	dead time in 1ms
I {current}	INT	average current in pA
Q {charge}	INT	charge in pC
$DETECTOR:		
ANGLE {angle}	REAL	detector angle in degrees
$PREP_POS:		position and orientation of the sample
X {x}	REAL	
Y {y}	REAL	
Z {z}	REAL	
T {tilt}	REAL	
S {spin}	REAL	
R {rotation}	REAL	
$SCAN:		
NX {#pointsx}	INT	number of scan pattern points in the x-direction (1 or more)
NY {#pointsy}	INT	number of scan pattern points in the y-direction (1 or more)
DX {#stepsx}	INT	distance between the points in the x-direction (in DAC values)
DY {#stepsy}	INT	distance between the points in the y-direction (in DAC values)
OX {offsetx}	INT	offset in the x-direction (in DAC-values)
OY {offsety}	INT	offset in the y-direction (in DAC-values)
$DECODER:		
0 {data}	INT	data for the channeling decoder
.		
.		
7 {data}	INT	
$ROI:		info about a possible region of interest (ROI)
BEGIN {chan}	INT	start channel of the ROI
END {chan}	INT	end channel of the ROI
CNTS {roicounts}	INT	number of counts in the ROII
$DATA:		the actual histogram data (AXIL compatible)
{beginch} {nrch} follow	INT INT	start channel of the data and the number of channels to
{data}	INT	value for the start channel
.		
.		
{data}	INT	value for the last channel
$LAST_FILE:		
{last}	INT	0 = more files will follow, 1 = this is the last file in the series
$SUM_SPEC:		
{sum}	INT	0 = update , 1 = sumspectrum of the series up to now
$END:		end of file indicator

Appendix D
The user interface module CUI2

D.1 Structures

- Entity info structure

```
typedef struct {
   int dummy;
   int nNumFields;                  /* number of elements of .wdFields[] */
   int nCurrentId;                  /* id of the record currently edited by the entity dialog box */
   int nEntity;                     /* entity dialog box id as used in UIL file(s) */
   Widget wdShell;                  /* popup shell widget used by the entity dialog box */
   Widget wdNamesListBox;           /* the names list box of the entity */
   Widget wdFields[1];              /* widgets of the entity fields. Allocate memory when needed. */
   } EntityWidgetInfoRec, *EntityWidgetInfo;
```

- Combo box info structure

```
typedef struct {
   int dummy;
   Widget wdLabel;                  /* the Label widget of the combo box */
   Widget wdDropButton;             /* the "Drop" Buton widget of the combo box */
   Widget wdGotoButton;             /* the "Goto" Button widget of the combo box */
   Widget wdListBox;                /* the ListBox widget of the combo box */
   int nComboBox;                   /* the combo box id as used in UIL file(s) */
   int nReferedId;                  /* the id of the item that it refered to */
   } ComboBoxWidgetInfoRec, *ComboBoxWidgetInfo;
```

- List box info structure

```
typedef struct {
   int dummy;
   int nNumItems;                   /* number of items contained in the list box */
   int nCrossRef;                   /* an identifier of the cross-ref table to use */
   Widget wdComboBox;               /* the Widget of the combo box that the ListBox belongs to */
   int nSelected;                   /* the currently selected item in the list */
   int nIds[5];                     /* the secondary entity id's of which the names are listed in the list box */
                                    /* may also contain one or more integers for additional data fields in a */
                                    /* structure, like the par. sequence nr. in the par_in_module crossref. */
   } ListBoxWidgetInfoRec, *ListBoxWidgetInfo;
```

- Cross reference info structure

```
typedef struct {
   int dummy;
   int nNumItems;                   /* number of items contained in the list box */
   int nCrossRef;                   /* an identifier on the cross-ref table to use */
   Widget wdComboBox;               /* for compatibility only */
   int nSelected;                   /* the currently selected item in the list */
   int nIds[5];                     /* the secondary entity id's of which the names are listed in the list box */
                                    /* may also contain one or more integers for additional data fields in a */
                                    /* structure, like the par. sequence nr. in the par_in_module crossref. */
   } CrossRefWidgetInfoRec, *CrossRefWidgetInfo;
```

The cross refererence info structure and the list box info structure are used interchangeably. Therefore the size and contents of these structures should be the same!

D.2 Routines

- **int main(unsigned *argc*, char ***argv*)**
  Program entry point. Calls Initialize( ), COMMSetLogWindow( ) and XtMainLoop( ).
- **void UIMessage(char *sMessage, void *vExtra)**
  Displays a message dialog box with the message sMessage. vExtra may be an extra variable for use in formatted strings. The XmNokCallback function is set to DestroyMessageBox.
- **void UIBeep()**
  Gives a beep.
- **int UIChoice(char *sMessage, void *vExtra, int bAlsoCancel)**
  Displays a message box with the question sMessage. vExtra may be an extra parameter for use in formatted strings. The question box contains a YES and NO button. If bAlsoCancel=TRUE then a cancel button is added. Via the callback function CautionAnswer the user's choice is returned (CC_YES/CC_NO/CC_CANCEL)
- **void UISetState(char *sState, void *vExtra)**
  Writes the string sState to the status line. vExtra can be used for additional arguments.
- **void UIGetOut(int *nErrorLevel*)**
  Quits the program after aborting monitoring, destroying all graphics, closeing the data base and terminating communication with PhyDAS. nErrorLevel is returned.
- **int UIShowCommand(int *bShow*)**
  Shows/hides the command line window uppon bShow. The height of the command line window is returned.
- **void UIProcessPendingEvents()**
  Processes all pending events.
- **static void Initialize(unsigned *pargc, char **argv)**
  Initialisation: Error handling, application shell creation, interpretation of command line parameters (in argv), connection to the Oracle data base, UIL resource file is opened and the widget hierarchy is fetched, main window and top level window creation
- **static int NonFatalErrorHandler(Display *display, XErrorEvent *errorEvent)**
  Non-fatal error handler.
- **static int FatalErrorHandler(Display *display)**
  Fatal error handler.
- **static Widget FetchNewWindow(char *sWdIndex, Widget *wdParent*)**
  The window called sWdIndex is fetch from the resource file and managed as a child of widget wdParent. The widget handle of the window is returned.
- **static void MakeEntity(int *nEntity*, int *nFirstId*)**
  Fetches the window for the entity nEntity as a child of the main window. The entity info structure is filled and the entity fields are filled by data from the data base for the id nFirstId.
- **static void FillNamesListBox(Widget *wdListBox*, int *nId*, char *sName)**
  Callback function called back by the data base routines for every entry in the listbox. The string sName is added to the list box with widget handle wdListBOx and the corresponding id nId is added to the listbox structure.
- **static void fFillGraphParListBox(struct stRowType *pRow)**
  Callback function on DBGetRow(). Fills the Module parameter listbox in the Graphic entity window.
- **static void MenuActivate(XmPushButtonGadget *wd*, int *pnMenuItem, unsigned long *reason*)**
  Callback function for push buttons from the menu.
- **static void CreateEntity(XmPushButtonGadget *wd*, int *pnEntity, unsigned long *reason*)**
  Callback function for creating a new entity. Calls MakeEntity().
- **static void CreateField(Widget *wd*, int *pbEntityIsGrandParent, unsigned long *reason)**

Callback function on every field in an entity. Creates the entity info structure if needed, and adds the field widget to thee field widget list in the entity info structure. The number of fields is also updated.

- **static void CreateCrossRef(Widget *wd*, int **pnCrossRef*, unsigned long *reason*)**
  (Re)allocated memory for the cross reference info structure. Some info is written to this structure, like the listbox that corresponds to this cross reference. Furthermore the cross reference is added as a field to the entity that it belongs to.
- **static void DialogGotFocus(Widget *wd*, char **sText*, unsigned long *reason*)**
  Callback function for updating the state line text.
- **static void CreateOkCancel(XmBulletinBoardWidget *wd*, int **pbVertical*, unsigned long *reason*)**
  Callback for creating the OK and Cancel button window in an entity window. The window is added to the entity info structure as field nr. 0.
- **static void PushOkCancelButton(XmPushButtonWidget *wd*, int *bOk*, unsigned long *reason*)**
  Callback function, called upon pressing Ok or Cancel in an entity window. The entity window is unmanaged, and when the Ok button was pressed, the data from the entity fields is save to the data base (after user confirmation) by calling UIDBSaveEntity().
- **static void DestroyMessageBox(XmMessageBoxWidget *wd*, XmMessageBoxWidget *wdMessage*, unsigned long *reason*)**
  Callback function on pressing Ok in the message box dialog. The message box is destroyed.
- **static void CautionAnswer(XmMessageBoxWidget *wd*, int *nAnswer*, unsigned long *reason*)**
  Callback function that handles the return of the answer on the caution box dialog.
- **static void ComboBoxDrop(XmPushButtonWidget *wd*, int nDummy, unsigned long **reason*)**
  Callback function called on pressing a drop button of a combo box. The drop list box is created and filled or destroyed is already existing.
- **static void ComboBoxCreate(Widget *wd*, int **pnComboBox*, unsigned long **reason*)**
  Callback function for creating a combo box. A combo box consists of a text field, a drop button and possibly a goto button. A combobox info structure is created together with a field entry in the entities info structure.
- **static void ComboBoxGoto(Widget *wd*, int *nTargetDialogBox*, unsigned long *reason*)**
  Called upon pressing the goto button of a combo box. The corresponding entity is created.
- **static void PushButton(XmPushButtonWidget *wd*, int **pnButton*, unsigned long *reason*)**
  Callback function for push buttons.
- **static void ToggleButton(XmToggleButtonWidget *wd*, int **pnButton*, unsigned long *reason*)**
  Callback function for toggle buttons
- **static void CreateWindow(Widget *wd*, int **pnWindow*, unsigned long **reason*)**
  Callback function for the creation of general windows, like the state line, command line work window and several list boxes.
- **static void CompTypesSelect(XmListWidget *wd*, int *nDummy*, XmListCallbackStruct **stList*)**
  Callback function called on selecting an item from the list of predefined component types.
- **static void ParTypesSelect(XmListWIdget *wd*, int *nDummy*, XmListCallbackStruct **stList*)**
  Callback function called on selecting an item from the list of predefined parameter types.
- **static void CrossRefSelect(XmListWidget *wd*, int *nDummy*, XmListCallbackStruct '**stList*)**
  Callback function called on selecting an item from the listbox corresponding to a cross reference.

- **static void ComboListBoxSelect(XmListWidget *wd*, int *nDummy*, XmListCallbackStruct **stList*)**
  Callback function called on selecting an item from the combo box list.
- **static void NamesListBoxSelect(XmListWidget *wd*, int *nDummy*, XmListCallbackStruct **stList*)**
  Callback function called on selecting an item from the names list box.
- **static void CommandEntered(Widget *wd*, int *nDummy*, XmCommandCallbackStruct **stCommand*)**
  Callback function called on entering a command at the command line.
- **static void StartUpSort(void)**
  Calls the list mode data sorting program SORTLMDATA with data from the data base.
- **void StartUpCalibration(int *nSerId*)**
  Starts the off-line analysis program Calibration Egg with data from the data base.
- **void StartUpAnalysis(int *nSerId*)**
  Starts the off-line analysis program Analyssi Egg with data from the data base.
- **Dimension Width_of_Widget(Widget *wd*)**
  Returns the width in pixels of a widget.
- **Dimension Height_of_Widget(Widget *wd*)**
  Returns the height in pixels of a widget.
- **Position XPos_of_Widget(Widget *wd*)**
  Returns the x position of a widget in pixels.
- **Position YPos_of_Widget(Widget *wd*)**
  Returns the y position of a widget in pixels.

Appendix E
The data base module CUIDB2

E.1 Structures

- Field info structure

```
typedef struct {
   char *sFieldName;              /* the field name as known by the database */
   int nType;                     /* the type of the field */
   } FieldInfoRec, *FieldInfo;
```

E.2 Variables

- FieldInfo *stEntityFields[]*
  The field info structures for every entity.
- char **sTables[]*
  The table names as they are passed to the data base.
- char **sEntities[]*
  The names of the entity dialogs in the UIL file.
- FieldInfo *stCrossRefFields[]*
  Array containing the cross reference field info structures for all cross references.
- char **sCRTables[]*
  The cross reference table names as they are passed to the data base.
- int *nCRNumFields[]*
  The number of fields for every cross reference. Usually equal to 2.
- int *nCREntities[]*
  The entity dialog in which the cross references can be edited.
- int *nComboBoxGoto[]*
  The entity of which the dialog should be created when pressing a goto button
- char **sComponentTypes[]*
  The predefined list of available components
- int *nComponentTypes*
  The number of predefined components
- char **sParameterTypes[]*
  The predefined list of the parameter types
- int *nParameterTypes*
  The number of predefined parameter types

E.3 Routines

- **void fFillstRow(struct stRowType *pRow)**
  Callback function used by UIDBLoadEntity for a DBGetRow call
- **void UIDBLoadEntity(XmBulletingBoardWidget wdEntity, int bNew)**
  Fill a the fields of an entity dialog with data from the data base. If bNew=TRUE then default values are provided.
- **int UIDBSaveEntity(XmBulletinBoardWidget wdEntity, int bReplace)**
  Saves the contents of the entity dialog fields to the data base. If bReplace=TRUE an existing record will be replaced.
- **void UIDBFillListBox(XmListWidget wdListBox, int nId, char *sName)**
  Callback function by DBGetWholeColumn() in order to add an item to a list box.
- **static void FillCrossRef(struct stRowType *pRow)**
  Callback function by DBGetRow() in order to add an item to a cross reference list box.
- **void FillNoFieldListBox(XmListWidget wdListBox, int nId, char *sReferedId)**
  Callback function by DBGetWholeColumn() for adding an entry to a general list box.

Appendix F
The monitoring module CMON2

F.1 Structures

- File Parameter info structure

```
typedef struct {
   int nMemoryMode;                /* 16 for 1 parameter in this file, 32 for 2 parameters in this file */
   int nFirstParId;                /* the id of the parameter */
   char sFirstParType[DB_VALUELENGTH];
                                   /* the type op the parameter (Energy, PIN, Equation etc.) */
   int nSecondParId;               /* the id of the parameter */
   char sSecondParType[DB_VALUELENGTH];
                                   /* the type op the parameter (Energy, PIN, Equation etc.) */
   int nParSeqNr;                  /* the parameter filenr. used to create te file name */
   FILE *fFile;                    /* the file structure pointer to the file where this parameter */
                                   /* can be retrieved from */
   unsigned short bufferFirst[MON_MAXBUFSIZE];
                                   /* buffer for the first parameter in the file */
   unsigned short bufferSecond[MON_MAXBUFSIZE];
                                   /* buffer for the last parameter in this file */
   } FileParameterInfoRec, *FileParameterInfo;
```

- Parameter window info structure

```
typedef struct {
   int nParId;                     /* the id of the parameter */
   unsigned short *nParBuffer;     /* pointer to the correct buffer in the fileparameterinfo for the par-id */
   int nLowerLimit;                /* lower limit value for this parameter */
   int nUpperLimit;                /* upper limit value for this parameter */
   } ParameterWindowInfoRec, *ParameterWindowInfo;
```

- Module info structure

```
typedef struct {
   int nModuleId;;                 /* the module id */
   char sPath[MFIL_PATHLENGTH];    /* the path where the fileseries are located */
   int nFirstIndex;                /* the file nr. of the first file in the series */
   int nLastIndex;                 /* the file nr. of the last file in the series */
   char sModuleChar[MFIL_PATHLENGTH];
                                   /* the module identification character */
   char sVersionChar[MFIL_PATHLENGTH];
                                   /* the version identification character */
   int nFileType;                  /* indicates if we're dealing with listmode or histogrammode fileseries */
   int nCurFileNumber;             /* the currently opened file number */
   int nBufferSize;                /* actual size of the read buffers */
   int nBufferIndex;               /* current index in the buffer */
   int nDataIndex;                 /* current index in the list mode data series */
   int nFileLength;                /* the number of events in the current file */
   int nCounts;                    /* the total number of counts that have come from this module */
   int nCurrent;                   /* the average current while the currently opened file has  been written */
   int nCharge;                    /* the charge (in pC) that has been cumulated till now */
   int bFinished;                  /* boolean: no more data coming from this module */
   int bLastFile;                  /* boolean: the currently read file is the last one */
   int bWaiting;                   /* boolean: waiting for phydas to deliver new data (write new files) */
```

```
  int nCloturingGraphic;          /* the last graphic using this module: the one who steps to next event */
  int nNumFiles;                  /* the number of parameters for this module */
  int nNumPars;                   /* the number of parameters for this module */
  FileParameterInfo *stFileParameter;
                                  /* the structure containing the info about the parameters in this module */
  } ModuleInfoRec, *ModuleInfo;
```

- **Scan position structure**

```
typedef struct {
  unsigned short x;               /* the DAC value for horizontal positioning of the scan magnet */
  unsigned short y;               /* the DAC value for horizontal positioning of the scan magnet */
  } ScanPosition, *ScanPositions;
```

- **Graphic info structure**

```
typedef struct {
  int nGraphId;                   /* the id of the graphic */
  int nXParId;                    /* the id of the parameter to be plotted along the x-axis */
  unsigned short *nXParBuffer;    /* pointer to the correct buffer in the fileparameterinfo for the x-par-id */
  int nYParId;                    /* the id of the parameter to be plotted along the y-axis (if necessary) */
  unsigned short *nYParBuffer;    /* pointer to the correct buffer in the fileparameterinfo for the y-par-id */
  int nType;                      /* indicates if it is a 1 parameter of 2 parameter graphic */
  int nNormBoolean;               /* indicates if the graphic is to be normalized */
  int nNormParId;                 /* id of the parameter that is to be used to normalize the graphic */
  unsigned short *nNormParBuffer;
                                  /* pointer to the buffer in the fileparameterinfo for the norm-par-id */
  int nNumPars;                   /* the number of parameter limitations for this graphic */
  ParameterWindowInfo *stParameterWindow;
                                  /* the structure containing info about windows on parameters in this module */
  ModuleInfo stModule;            /* the module used by this graphic */
  Widget wd;                      /* the GraphicWindow widget */
  Widget wdDialog;                /* the GraphicWindow's parent widget */
  Widget wdControlMenu;           /* the GraphicWindow's control menu */
  char sTitle[MON_TITLELENGTH];   /* the graphic's title */
  int nFromX;                     /* the start channel of the x-axis */
  int nToX;                       /* the end channel of the x-axis */
  int nFromY;                     /* the start channel of the y-axis */
  int nToY;                       /* the end channel of the y-axis */
  int nFromCounts;                /* the start of the counts axis */
  int nToCounts;                  /* the end of the counts axis */
  int nXInterval;                 /* the number of channels on the x-axis */
  int nYInterval;                 /* the number of channels on the y-axis */
  int nCountsInterval;            /* the length of the counts axis */
  int nWidth;                     /* the width of the graphic in pixels */
  int nHeight;                    /* the height of the graphic in pixels */
  int nXOrigin;                   /* the x coord of the origin in pixels */
  int nYOrigin;                   /* the y coord of the origin in pixels */
  float nXScale;                  /* the scale of the x-axis */
  float nYScale;                  /* the scale of the y-axis */
  float nLinCountsScale;          /* the scale of the linear counts-axis */
  int nLogCountsScale;            /* the scale of the log counts-axis */
  int nMonitoredCounts;           /* the number of counts monitored in this graphic */
  int nMaxCounts;                 /* the maximum number of counts int this graphic */
  int nMinCounts;                 /* the minimum number of counts in this graphic */
  int bLog;                       /* logaritmic scale for counts if true, else linear */
  int *nLogScale;                 /* XtMalloc: length=(graphic height or num. colors). */
                                  /* nLogScale[i]=10^(i*nYScale/length) */
```

```
  int *nData;                    /* XtMalloc: the number of counts for every data point */
  unsigned short *nLogData;      /* XtMalloc: nLogScale[nLogData[i]]-nData[i]. */
                                 /* Bit 15: boolean: value changed */
  } GraphicInfoRec, *GraphicInfo;
```

- **File header structure**

```
typedef struct {
  unsigned short nHeaderLength;     /* the length of the file header (in bytes) */
  unsigned short nDataLengthLow;    /* least significant part of the length of the data block (in bytes) */
  unsigned short nMemoryMode;       /* the memory mode (16 or 32)*/
  unsigned short bLastFile;         /* boolean: last file of the series */
  unsigned short nChargeHigh;       /* most significant part of the charge (in pC) */
  unsigned short nChargeLow;        /* less significant part of the charge (in pC) */
  unsigned short nTimeLow;          /* time (in 0.01s) it has taken to measue the data written in the file file */
  unsigned short nDeadTime;         /* dead time of the MCA's */
  unsigned short nDataLengthHigh;   /* most significant part of the length of the data block (in bytes) */
  unsigned short nTimeHigh;
  } FileHeaderRec, *FileHeader;
```

F.2 Variables

- static GraphicInfo **stGraphics*
  Array of pointers to the graphic info structures for all current graphics.
- static int *nNumGraphics*
  The current number of graphics.
- static ModuleInfo **stModules*
  Array of pointers to the module info structures for all current modules.
- static int *nNumModules*
  The current number of modules.

F.3 Routines

- **void MONStop()**
  Stops monitoring as soon as possible. (Sets bStopped to TRUE)
- **int MONContinue()**
  Continues monitoring after it was stopped by the user. Returns FALSE if the monitoring could not be continued
- **int MONBusy()**
  Returns TRUE is monitoring is busy.
- **void MONClearUp()**
  Destroys all graphic windows and frees the memory associated with them.
- **void MONSetSpeed(int nSpeed1, int nSpeed2, int nSpeed3)**
  Changes the monitoring speed (at once): update period, lost events and read buffer size.
- **void MONGetSpeed(int *pnSpeed1, int *pnSpeed2, int *pnSpeed3)**
  Returns the parameters associated with the current monitoring speed.
- **void MONCascade()**
  Resizes and moves all graphic windows into cascade.
- **void MONTile()**
  Resizes and tiles all graphic windows.
- **int MONStart(Widget wdMonit, int nExpId, int bMultiColor, char *sMoncolors)**
  Handles the list mode monitoring for one experiment. Returns TRUE/FALSE upon success/failure.
- **static void EchoGetScanPattern(char *sEcho)**
  Callback function for obtaining the scan pattern coming from PhyDAS.
- **static int MakeStandardPattern(int nStandardNr)**
  Creates a scan pattern based on the provided standard number.
- **static void fFillGraphic(struct stRowType *pRow)**

Callback function by DBGetRow for each graphic with the correct module. The row structure contains the id of the monitoring demand and the id of the graphic. Allocates memory for storing the graphic info structure and fills the structure. If needed the memory for the corresponding module's info structure is also allocated and that structure is filled also.

- **static void fFillParameterWindow(struct stRowType *pRow)**
  Callback function by DBGetRow() for each parameter with the correct module. The row structure contains the id of the graphic, the id of the parameter and the lower and upper limit for this parameter's value. If needed memory is allocated for storing the parameter window info structure and the structure is filled.
- **static void fFillModuleParameter(struct stRowType *pRow)**
  Callback function by DBGetRow() for every parameter with the correct module. The row structure contains the id of the module, the id of the parameter, the parameter sequence number and the parameter sub sequence number. If needed memory is allocated for the file parameter info structure and it is filled.
- **static void fFillModuleSeries(struct stRowType *pRow)**
  Callback function by DBGetRow() for each fileseries with the correct experiment id, module id and filetype. The row structure contains: the id of the file series, the id of the module, the path name of the file series the first and last file number of the file series, the module and version identification character and the file type. This data is written to the module info structure.
- **static void MainMonitoringLoop()**
  The main loop for working out the list mode monitoring
- **static int OpenFiles(ModuleInfo stModule)**
  Opens all files for the module. Then reads the headers and return TRUE/FALSE upon succes/failure.
- **static void CloseFiles(ModuleInfo stModule)**
  Closes all files for the module
- **static int ReadFiles(ModuleInfo stModule)**
  Reads data from the files via buffers. Returns FALSE on end of file.
- **static int ReadBuffers(ModuleInfo stModule, int nBeginAt, int nStep)**
  Reads an amount of data from all module files into buffers. PEP integers are converted into VMS short integers. Returns TRUE/FALSE upon success/failure.
- **static void UpdateGraphic(GraphicInfo stGraph, int bDrawAll)**
  Updates a graphic based upon the graphic type.
- **static void GraphicExposed(Widget wd, GraphicInfo stGraph, XmDrawingAreaCallbackStruct *callBack)**
  Callback function on a graphic expose event. Redraws the graphic.
- **static void RedrawGraphic(Widget wd, GraphicInfo stGraph, unsigned long *reason)**
  Redraws the whole graphic.
- **static void DestroyGraphic(Widget wd, Graphic stGraph, unsigned long *reason)**
  Callback function called when a graphic window is closed or destroyed.
- **static void EnableDragAndControl(Widget wd)**
  Enables the dragging and raising of windows and other mouse actions.
- **static void TrackHandler(Widget wd, Opaque closure, XMotionEvent *eventP)**
  Handles the Button2Motion events. Displays the coordinates of the mouse position within the graphic.
- **static void ButtonPressHandler(Widget wd, Widget wdGraph, XButtonEvent *eventP)**
  Handles the ButtonPressHandler events for zooming, contour definitions and control menu actions.
- **static Widget CreateControlMenu(GraphicInfo stCurGraph)**
  Creates a control menu for this graphic that is popped up op MB3 press. The widget of the created control menu is returned.

- **static void ActivateControlMenu(XmPushButtonWidget wd, int nItem, unsigned long *reason)**
  Callback function for handling control menu selections.
- **static void RecalculateScales(Widget wd, GraphicInfo stGraph, unsigned long *reason)**
  Recalculates the X and Y axis scales, based on the current width and height of the graphic.
- **static void RecalculateCountsScale(GraphicInfo stGraph, int nMaxCounts)**
  Recalculates the countsscale of the graphic, based on the height or the number of colours of the graphic.
- **static int RecalculateLogScale(GraphicInfo stGraph)**
  (Re)allocates memory for storing the pre-calculated values of the logarithmic scale of the graphic.
- **static int FloatToInt(double x)**
  Float to Integer type conversion function
- **static unsigned short PEPToVMS(unsigned short x)**
  PEP to VMS type conversion function
- **static void WaitingUIMessage(char *sFileName)**
  Displays the Waiting for PhyDAS message using the same message box, so the user doesn't have to click every time this message appears.
- **static void DestroyWaitingBox(Widget wd, Widget wdWait, unsigned long *reason)**
  Callback function for destroying the waiting message box.
- **static void PrintWindow(Widget wd)**
  Dumps the contents of the refered window into a HP Paintjet file.
- **static void SaveWindow(Widget wd)**
  Saves the contents of the refered window into a Windows 3.0 bitmap file.

Appendix G
The Columbus data base entities

In this appendix all entities from the data base are discussed. The purpose of every entity and its fields are discussed and an overview of the user interface for editing the data base if given. The data dictionary of the data base can be found in [GER94].

G.1 The experiment entity

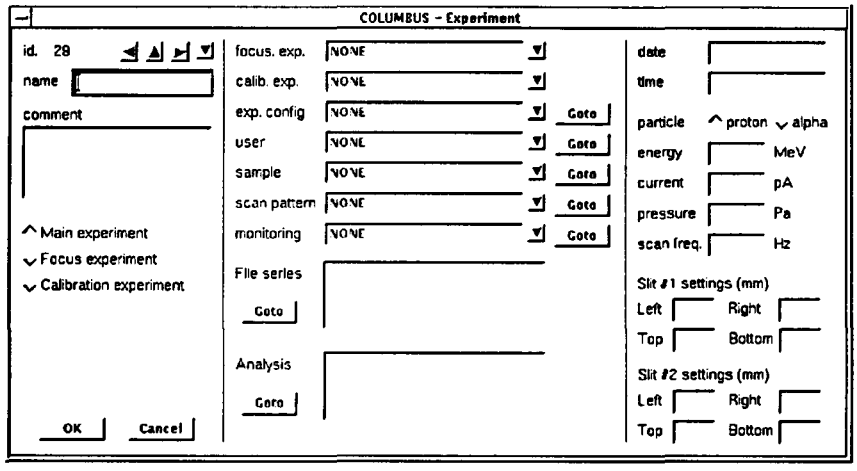

Figure G.1: The experiment entity dialog

The experiment entity is the central entity in the Columbus data base, as can be seen in the data base model. It has links to most of the other entities in the data base. It is the starting point for adding a new experiment to the data base.
Every entity has a record id. field and a name field in the upper left corner. Buttons are available for selecting an existing entity record and for adding a new record. Usually a comment text field is available for adding additional comment to an entity record.
The type of the experiment can be selected via toggle buttons. Focus experiments are currently only done with the micro beam experiment. A calibration experiment may be an experiment for calibrating the energy signals and all other experiments are of the type main experiment.
A measuring session consists of several parts. If using the microbeam set-up first of all a focus experiment is performed for minimizing the beam spot size on the sample. Then a calibration experiment may be done before the other main experiments are started.
Via the focus experiment combo box in the experiment entity dialog box a reference can be set to the corresponding focus experiment for calibration and main experiments.
Via the calibration experiment combo box a reference is set to the corresponding calibration experiment for the main experiments.
The reference to the experiment configuration that was used for the experiment can be set via the experiment configuration combo box. The user combo box is used to specify the user that performed the experiment. Furthermore the sample that was used during

the experiment can be refered to. When using a scan pattern during the experiment (only for the microbeam set-up) this can be refered to via the scan pattern combo box.
The last reference that can be set is a reference to a monitoring demand entity record that contains the info about the graphics that have to be displayed during monitoring the experiment.
Two list boxes are available that contain a list of resp. the fileseries entity records and the analysis result entity records that refer to the experiment.
Finally the experiment entity dialog contains some fields that are not references to other entities but that contain general data, like the time and date of the experiment, beam information and slit settings.

G.2 The configuration entity

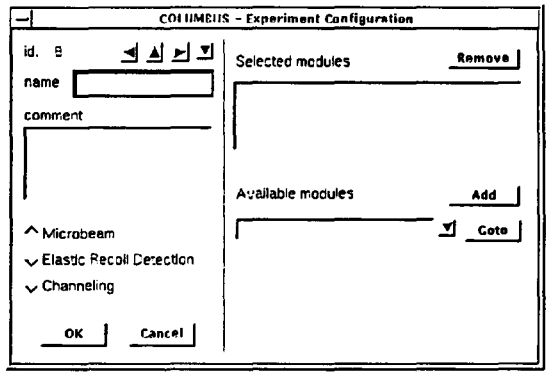

Figure G.2: Configuration entity dialog

In the configuration entity first of all the experiment set-up can be specified via a set of toggle buttons. Furthermore this entity dialog is used to edit the Module_in_Config cross reference for specifiying what modules are used in this experiment configuration.

G.3 The module entity

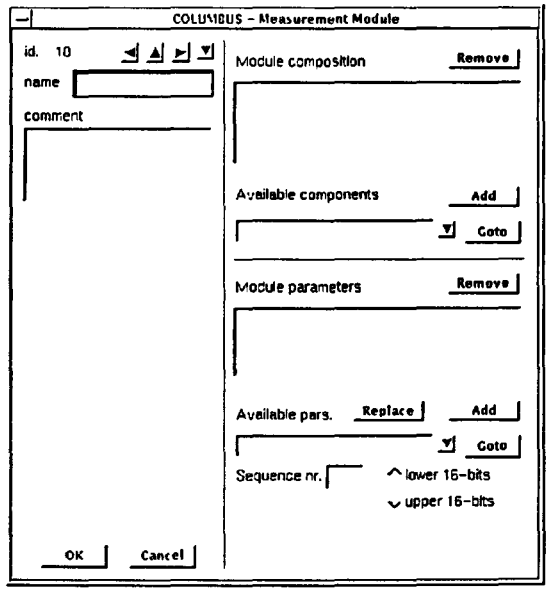

Figure G.3: The module entity dialog

Via the module entity dialog two cross reference entities can be edited; the Comp_in_Module cross reference for setting the hardware components of which a module consists and the Par_in_Module cross reference for setting the parameters that are contained in this module. For every parameter two extra specifications have to be set. First of all the sequence number of the parameter within this module. This corresponds to the sequence number within the module of the list mode memory in which the parameter's data is stored during list mode experiments of the sequence number within the module of the histogram mode memory in which the histogram for the parameter is collected during histogram mode experiment. Because list mode memories can contain two parameters a sub-sequence number is indicated if the parameter is the first or the second parameter within the memory.

G.4 The module component entity

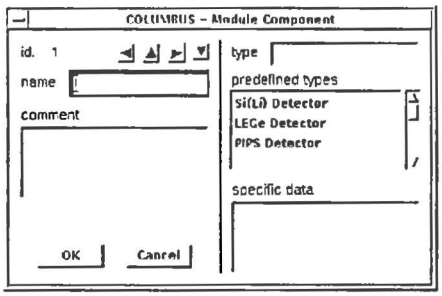

Figure G.4: The module component entity

This entity is used for defining a module component. The type of the component can be selected from a list of predefined types. A specific data field is added for future use purposes.

G.5 The module parameter entity

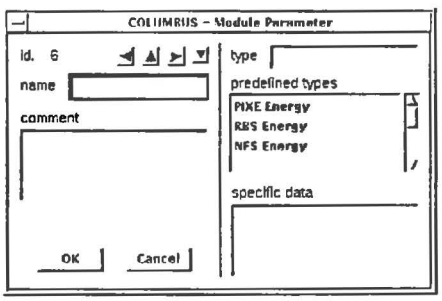

Figure G.5: The module parameter entity

This entity is used for defining a module parameter. The type of the parameter can be selected from a list of predefined types. A specific data field is added for future use purposes.

G.6 The file series entity

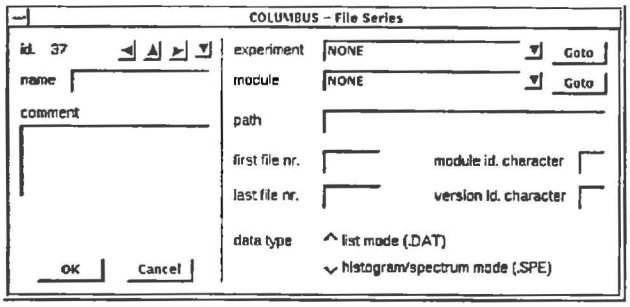

Figure G.6: The file series entity

A file series refers to both an experiment and a module. A files of a file series contain the data of all parameter in the corresponding module. Furthermore the entity contains fiels for specifying the directory path where the files of the series can be found, the first and the last file number of the series and a module identification and version identification character for creating the file names. The type of the data of the file series (list mode data or histogram mode data) can be specified via a set of toggle buttons.

G.7 The monitoring demand entity

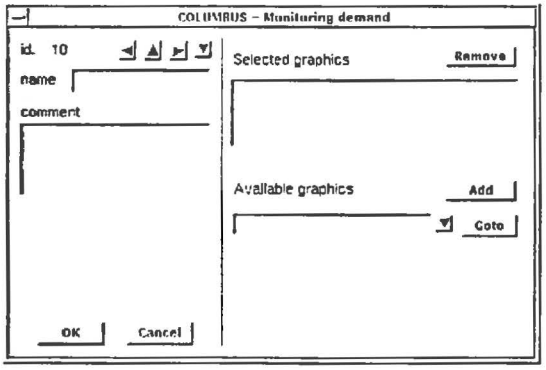

Figure G.7: The monitoring demand entity

Via the monitoring demand entity dialog the Graph_in_Mon cross reference can be edited. It is used to specify what graphics are to be displayed during monitoring.

G.8 The graphic entity

Via the graphic entity the layout of a graphic can be specified. First of all a reference to a module has to be set via which the available parameters can be retrieved. Then a reference to one or two parameters of which a histogram has to be created and shown. This can be a one parameter histogram with the parameter channel versus the yield or a two parameter scatterplot with the yield as a function of the channels of two parameter. Furthermore a reference can be set to a parameter that will be used for normalizing the graphic to the integral yield of this parameter.

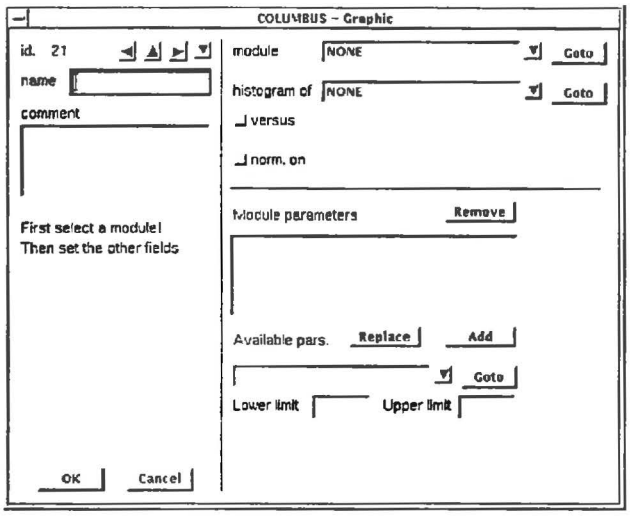

Figure G.8: The graphic entity

Finally the Par_in_Graph cross reference can be edited for setting conditions on parameters. A condition consists of a reference to a parameter and a minimum and maximum (integer) value for that parameter.

G.9 The sample entity

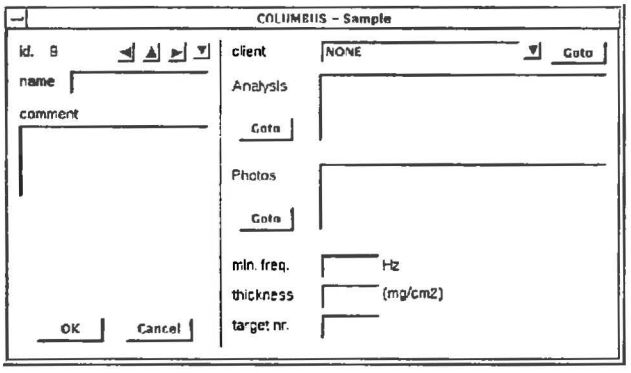

Figure G.9: The sample entity

The sample entity is used for storing info about a sample. A reference can be set to a client that provided the sample. Two listboxes are available containing an list of references from the analysis demand entity and the photo entity to this sample.
Data fields are available for specifying a minimum sample frequency that is needed to prevent destruction of the sample when using it in the microbeam set-up, for the mass thickness of the sample and for the number of the location of the sample within the target holder.

G.10 The user entity

For specifying a user. Together with a name the telephone number and the status of the user can be specified.

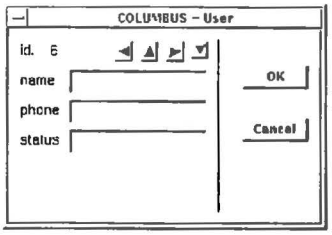

Figure G.10: The user entity

G.11 The client entity

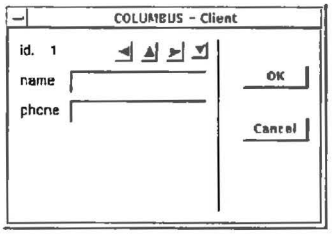

Figure G.11: The client entity

For specifying a client for whom analysis has to performed on a sample. Together with a name the phone number of the client can be specified.

G.12 The analysis demand entity

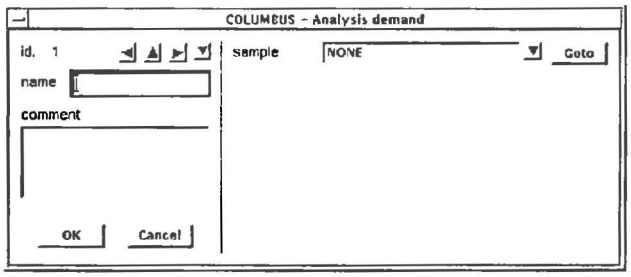

Figure G.12: The analysis demand entity

The analysis demand entity is for specifying a type of analysis that has to be performed on a sample. It is not yet fully specified, but it will have to contain a reference to a sample.

G.13 The analysis result entity

The analysis result entity is for storing the results of an analysis sequence into the data base. Although it is not yet fully specified if will at least have to contain a reference to an experiment and a module of which the data has been analysed. Most likely a reference to a file containing the analysis results will have to be added.

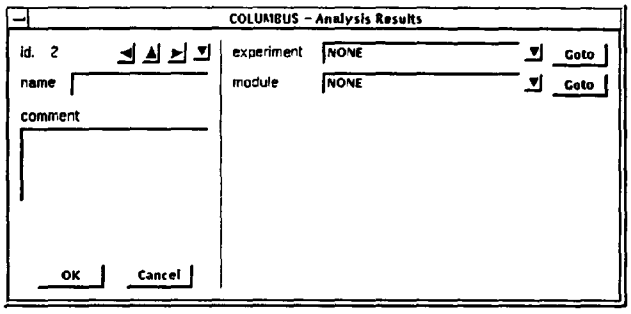

Figure G.13: The analysis result entity

G.14 The photo entity

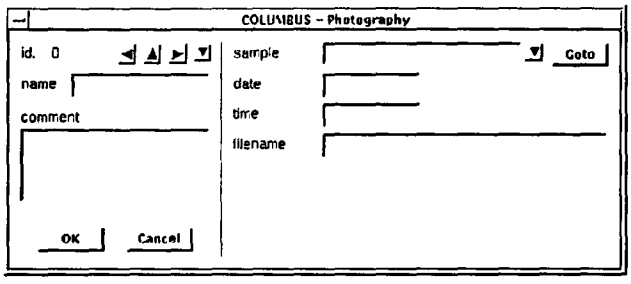

Figure G.14: The photo entity

The photo entity is used for storing info about a photograph that has been taken of a sample with a CCD camera and video frame grabber. It contains a reference to this sample and fields for specifying the date and time at which the photo was taken and the location of the picture file on disk.

G.15 The scan pattern entity

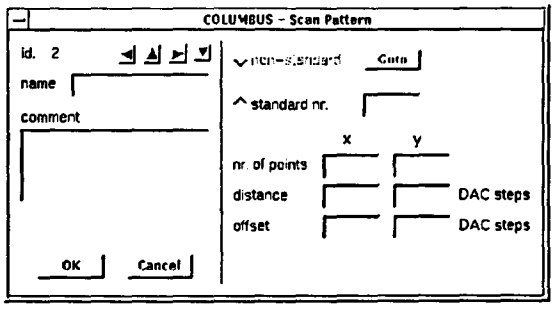

Figure G.15: The scan pattern entity

The scan pattern entity is for specifying the shape of a scan pattern. Only standard scan patterns are supported at the moment. For a standard scan pattern, together with the standard number, the horizontal and vertical number of points, the distance between these points and an offset has to be specified.