This is to certify that the
thesis entitled

PRODUCTION OF NUCLEI IN NEUTRON UNBOUND STATES VIA PRIMARY FRAGMENTATION OF ^{48}CA

presented by

GREGORY ARTHUR CHRISTIAN

has been accepted towards fulfillment
of the requirements for the

MASTER OF SCIENCE degree in PHYSICS AND ASTRONOMY

Major Professor's Signature

4-14-08

Date

MSU is an affirmative-action, equal-opportunity employer

PRODUCTION OF NUCLEI IN NEUTRON UNBOUND STATES VIA PRIMARY FRAGMENTATION OF ^{48}CA

By

Gregory Arthur Christian

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Physics and Astronomy

2008

ABSTRACT

PRODUCTION OF NEUTRON UNBOUND NUCLEI VIA PRIMARY FRAGMENTATION OF ^{48}CA

By

Gregory Arthur Christian

The method of sequential neutron decay spectroscopy beginning with a primary beam of 60 MeV/nucleon ^{48}Ca was investigated as a potential tool to measure unbound resonances in neutron rich nulei. Neutrons were measured in coincidence with fragments, and unbound resonances were observed for ^{10}Li, ^{12,13}Be, and ^{23}O. The method is currently limited to resonances with small decay energies ($E_{\mathrm{decay}} \lesssim 100$ keV) in lighter ($Z \leq 10$) nuclei, and the possibility of extending it to heavier, more neutron-rich nuclei will be discussed.

Contents

1 Introduction **1**
- 1.1 Structure of Neutron Rich Nuclei 1
- 1.2 Neutron Spectroscopy 2

2 Experimental Setup and Calibrations **7**
- 2.1 Sweeper Magnet 10
- 2.2 MoNA 13

3 Data Analysis **17**
- 3.1 Particle Identification 17
- 3.2 Scattered Background 23
- 3.3 Decay Energy Reconstruction 25
- 3.4 Monte Carlo Simulations 26

4 Results and Discussion **31**
- 4.1 Results 31
- 4.2 Analysis of Low Energy Decays 39

5 Future Prospectives **55**

6 Summary **58**

A Using the MoNA Simulation Code **59**
- A.1 Introduction 59
- A.2 Getting Set Up 60
- A.3 Running the Simulation 62
- A.4 Using Flags 64
  - A.4.1 Decay Energy Model 67
  - A.4.2 Stripping Reaction Model 67
  - A.4.3 GEANT Output 68
- A.5 Analyzing Your Simulation 69
  - A.5.1 Parameter Names 71

A.6 Customizing the Simulation 73
A.7 Using SVN 77

B Using ROOT **79**
B.1 Introduction 79
B.2 Installing ROOT 80
B.3 Files and Trees 81
B.4 Drawing Histograms 84
B.5 TCanvases 87
B.6 Gates 89
B.7 Creating Pseudo Parameters 91
B.8 Using Macros 92
B.9 Various Odds and Ends 94

Bibliography **97**

List of Tables

4.1 Summary of Previous ^{10}Li Ground State Measurements. 40

A.1 Simulation Beamline Element Numbering Scheme. 72

A.2 Simulation Parameter Naming Scheme. 72

B.1 Example ROOT TTree. 82

List of Figures

1.1 Chart of Light Neutron Rich Nuclei 2

1.2 SDNS Setup at Large Angle 5

1.3 Fragmentation Reaction Angular Distribution. 5

1.4 Sample Zero Degree SDNS Setup. 6

2.1 Beam Production. 8

2.2 Experimental Setup 9

2.3 TDC Pulser Calibration. 10

2.4 Masked CRDC Position Spectrum. 11

2.5 Sample MoNA Bar Time Difference Spectrum. 13

2.6 MoNA Angular Acceptance. 14

2.7 γ-ray ToF. 15

3.1 Uncorrected Particle ID. 18

3.2 Focal Plane x-vs-θ. 19

3.3 Particle ID x-correction factors. 20

3.4 Particle ID θ-correction factors. 20

3.5 Particle ID 21

3.6 MoNA ToF for ^{10}Be Fragment 22

3.7 Scattered Background 24

3.8 Tracking of Background Events Through the Sweeper. 25

3.9 ^{9}Li Fragment Energy Distribution. 26

3.10 Neutron Kinetic Energy Distributions. 27

3.11 Opening Angle Distribution. 28

4.1 Observed Isotopes 32

4.2 Velocity Difference Spectra 34

4.3 Decay Energy Spectra 35

4.4 Event Mixed Decay Energies. 36

4.5 Event Mixed Decay Energies. 37

4.6 Efficiency of the Experimental Setup 38

4.7 Simon *et al* ^{10}Li Decay. 41

4.8 ^{9}Li + n Decay Energy Spectrum 42

4.9 ^{9}Li and ^{10}Li Level Scheme. 43

4.10 Deák *et al* ^{10}Be + n Relative Velocity. 45

4.11 Peters *et al* ^{11}Be Decay. 45

4.12 ^{10}Be + n Decay Energy Spectrum 46

4.13 ^{10}Be and ^{11}Be Level Scheme. 47

4.14 Thoennessen *et al* ^{12}Be + n Relative Velocity. 49

4.15 ^{12}Be + n Decay Energy Spectrum 50

4.16 ^{12}Be and ^{13}Be Level Scheme. 51

4.17 Schiller *et al* ^{23}O Decay. 52

4.18 ^{22}O + n Decay Energy Spectrum 53

4.19 ^{22}O and ^{23}O Level Scheme. 54

5.1 Neutron ToF at Different Beam Rates 55

B.1 Example 1d ROOT Histograms. 85

B.2 Example 2d ROOT Histogram and Divided Canvas. 86

B.3 Example of 2d Gates in ROOT. 91

Images in this thesis are presented in color.

Chapter 1

Introduction

1.1 Structure of Neutron Rich Nuclei

Nuclei are highly complex, many body quantum mechanical systems consisting of spin 1/2 fermions (protons and neutrons) which interact via the strong nuclear and coulomb forces. The strong force is attractive and responsible for holding nuclei together, while the coulomb force provides a relatively small repulsive interaction between the positively charged protons.

Due to their complexity, phenomenological models must be employed to describe the structure of nuclei. The most widely used model is the shell model, which subjects the constituent protons and neutrons in a nucleus to a mean field potential. The form of this potential is adjusted to reproduce experimental observations. Typical shell model potentials build upon the Woods-Saxon potential with a spin-orbit coupling term [1]:

$$V(r) = \frac{-V_0}{1 + e^{(-r-R)/\alpha}} - V_{\ell s}. \tag{1.1}$$

Subjecting nucleons to a potential well causes them to occupy discrete energy levels. These energy levels are observed to group into shells, analogous to the case of electrons in an atom. The energy gaps within a shell are relatively small, and those between neighboring shells are much larger. Nucleons can be excited to higher energy

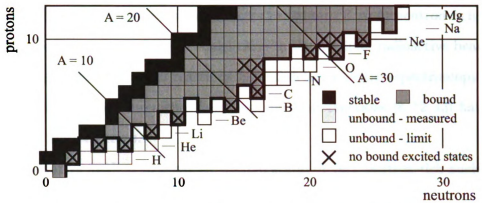


Figure 1.1: Chart of the nuclides showing neutron rich isotopes in the $Z \leq 12$ region. Unbound isotopes and isotopes without any bound excited states are indicated on the figure [2].

levels within the outermost shell or between shells if enough energy is added to the nucleus.

There is evidence that the shell structure of very neutron rich nuclei is significantly different from that of stable nuclei. Shell models describing nuclei in this region are often tested based on their reproduction of nuclear energy levels. For nuclei lying beyond the neutron dripline—those for which the number of neutrons is so great that the nuclei are unstable with respect to neutron emission—the energy of the ground state with respect to neighboring nuclei is also of interest.

1.2 Neutron Spectroscopy

The excitation spectra of ordinary nuclei are typically studied using γ-ray spectroscopy. This involves measuring the energy of the γ-rays emitted when an excited nucleus de-excites to a lower energy state. For very neutron rich nuclei, however, excited states often lie above the one-neutron separation energy, as indicated in Figure 1.1. The lack of bound excited states makes it difficult to study the structure of these nuclei with γ-ray spectroscopy. Thus other techniques must be employed to observe unbound excited states, whose decay is dominated by neutron emission.

Single and multiple particle transfer reactions from stable beams have been used

to populate and characterize unbound states in lighter ($Z \leq 4$) neutron-rich nuclei [3,4]. In order to populate heavier nuclei close to the dripline, radioactive beams are required. In addition to transfer reactions [5–8], β-delayed neutron spectroscopy [9–11] and invariant mass measurements following knockout reactions [8, 12, 13] have also utilized radioactive beams.

An alternative method to populate excited states of very neutron-rich nuclei with stable beams is sequential neutron decay spectroscopy (SNDS). In this method the excited nuclei are produced in heavy-ion reactions and the decay fragments are measured in coincidence with the emitted neutrons in a collinear geometry.

Early SDNS experiments detected neutrons and fragments at large angles relative to the beam [14–16]. An example setup, taken from Ref. [15], can be seen in Figure 1.2. In these early experiments, the angular separation between fragments and neutrons was needed to allow for event by event separation of neutrons and charged fragments. Large angle detection is disadvantageous, however, because medium energy fragmentation reactions are kinematically focused near zero degrees. Hence the yield for large angle fragment-neutron coincidences is rather low. This is demonstrated in Figure 1.3, which shows a simulated angular distribution, produced by the code LISE++ [17], for the fragmentation of ^{18}O into ^{9}Li. As can be seen in the figure, the relative yield peaks at zero degrees and falls off sharply at larger angles.

Later SDNS experiments, beginning with Kryger *et al.* [18], made use of a magnetic spectrometer to separate charged fragments from neutrons. In this configuration, the neutron detector can be placed at zero degrees relative to the incoming beam, resulting in a greater yield of detected coincidences than in the case of a large angle setup. A generic schematic of the zero degree setup is shown in Figure 1.4. The higher yield afforded by the zero degree setup allowed more neutron rich nuclei to be studied, including nuclei beyond the dripline [18–20].

Zero degree SDNS experiments performed in the past [18–20] have only measured two observables: neutron velocity (v_{n}) and fragment velocity (v_{f}). In order to extract

a decay energy from these measurements, the authors must rely on the approximation that the opening angle between the fragment and the neutron, θ_{open}, is equal to zero. This approximation is valid since the angular acceptance of these experiments is small. In the zero degree approximation, decay energy can be approximated as:

$$E_{\text{decay}} \simeq \mu \left(\frac{1}{\sqrt{1 - v_{\text{rel}}^2/c^2}} - 1 \right), \tag{1.2}$$

where μ is the reduced mass of the neutron-fragment system. In practice, experiments using the small angle approximation make use of Monte-Carlo simulations to relate E_{decay} to v_{rel}. These simulation programs model E_{decay} with a chosen distribution, then track the fragment and neutron through the experimental system, applying all relevant resolution and acceptance effects. A simulated relative velocity curve is then calculated and compared directly to the data.

Although the use of Equation 1.2 proves to be quite useful in extracting decay properties of unbound nuclei, it is desirable to make a measurement of E_{decay} directly, rather than rely on the approximation. This can be achieved by measuring θ_{open}, which makes it possible to calculate E_{decay} directly using the invariant mass method:

$$E_{\text{decay}} = \sqrt{m_{\text{f}}^2 + m_{\text{n}}^2 + 2\left(E_{\text{f}} \cdot E_{\text{n}} - p_{\text{f}} \cdot p_{\text{n}} \cdot \cos\left(\theta_{\text{open}}\right)\right)} - m_{\text{f}} - m_{\text{n}}. \tag{1.3}$$

Until now, the use of zero degree SDNS measurement techniques has been limited to the study of light nuclei ($Z \leq 4$) produced with light primary beams ($Z \sim 8$). Moreover, past SDNS experiments have relied on relative velocity measurements to estimate decay energy [18–20]. In the present work, the feasibility of using zero degree SDNS to extract a direct measurement of E_{decay} for heavier isotopes, starting with a primary beam of ^{48}Ca, is explored.

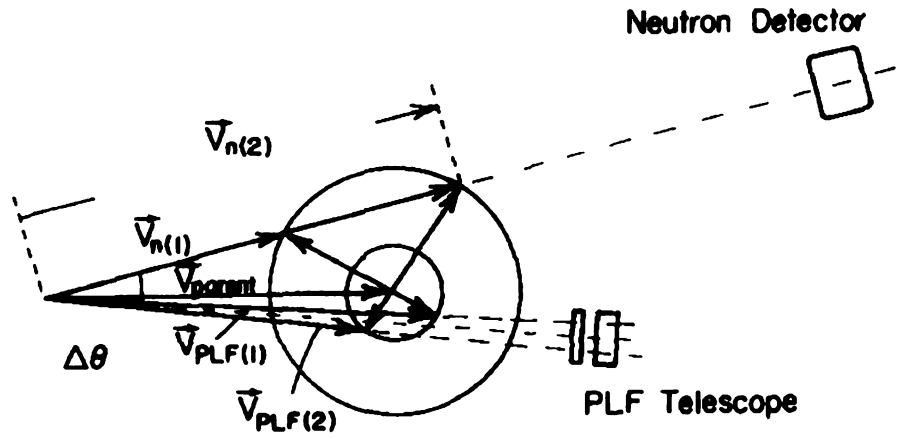


Figure 1.2: Example SDNS experimental setup at large angle, taken from Ref [15].

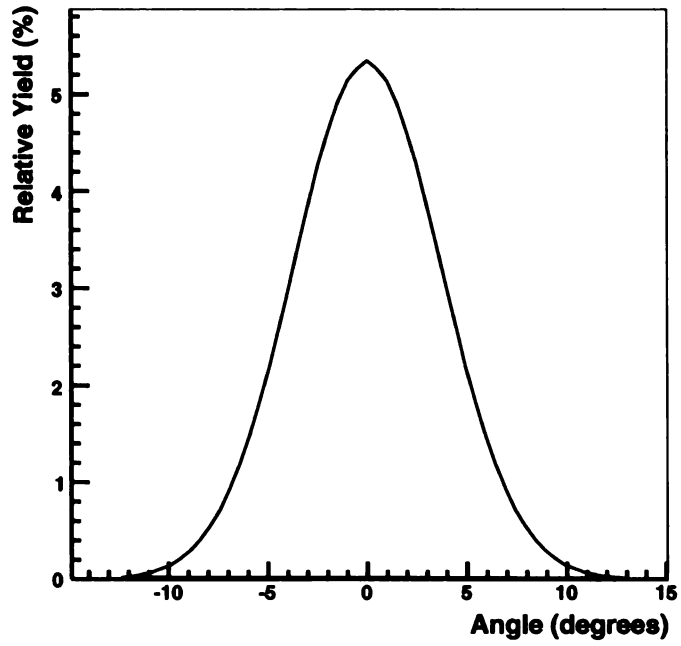


Figure 1.3: Simulated angular distribution for the projectile fragmentation reaction ^{9}Be(^{18}O, ^{9}Li)X. [17].

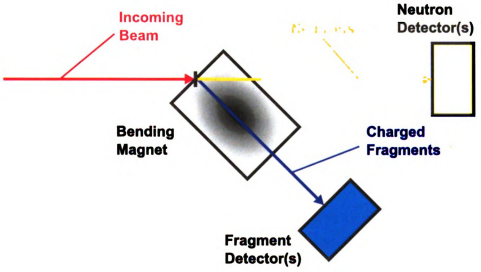


Figure 1.4: Generic SDNS setup with neutron detector at zero degrees.

Chapter 2

Experimental Setup and Calibrations

The experiment was performed at the National Superconducting Laboratory (NSCL) at Michigan State University. A primary beam of ^{48}Ca was accelerated to 90 MeV/u using the coupled K-500 and K-1200 cyclotrons [21]. In order to allow reaction products to be bent by the sweeper magnet (see below), the beam was degraded to 62.8 MeV/u in the A1900 fragment separator [22]. The reason for first accelerating the beam to 90 MeV/u then degrading it, as opposed to starting out with a $\sim$ 60 MeV/u beam, is that 90 MeV/u ^{48}Ca is a standard tune of the NSCL's coupled cyclotron system. Hence a higher beam quality is achieved by accelerating to 90 MeV/u then degrading, as opposed to accelerating directly to $\sim$ 60 MeV/u. After exiting the A1900, the beam was delivered to a 94 mg/cm^2 beryllium reaction target. A schematic of the beam production setup can be seen in Figure 2.1.

Charged fragments produced in the reaction of calcium on beryllium were first sent through a focusing quadrapole triplet and then deflected through a large gap sweeper magnet [23]. Neutrons produced in the reaction were detected by the Modular Neutron Array (MoNA) [24, 25] located at zero degrees. The timing start signal for all events was provided by the cyclotron radio frequency (RF) signal. A schematic of the experimental setup can be seen in Figure 2.2.

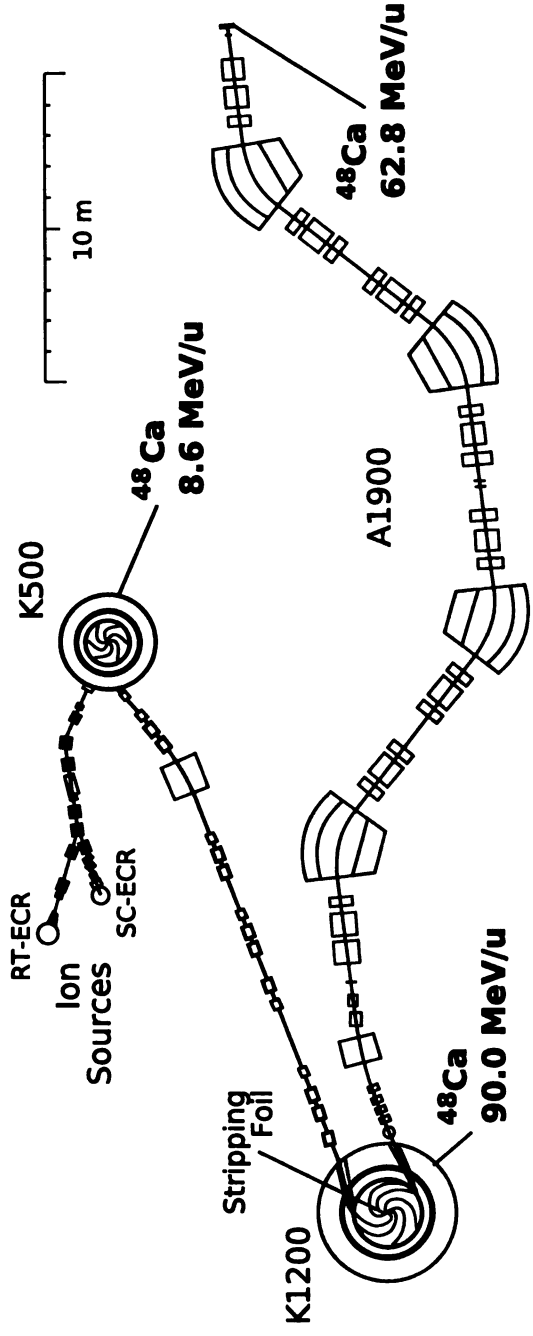


Figure 2.1: Schematic of the beam production setup.

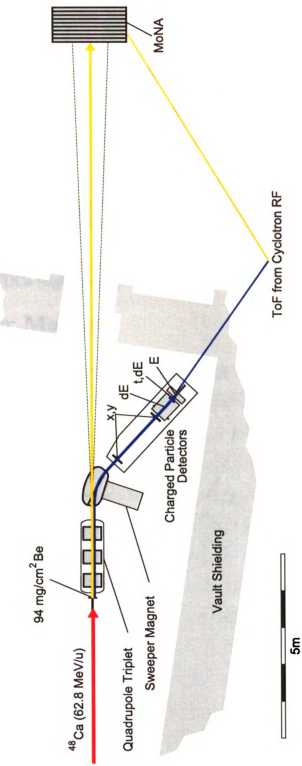


Figure 2.2: Experimental setup.

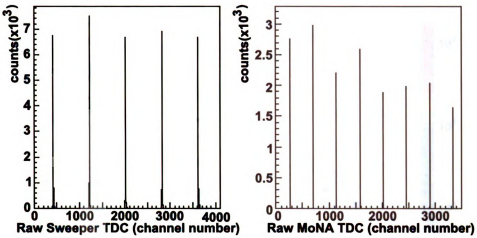


Figure 2.3: Example TDC spectra from pulser calibration runs for sweeper TDCs (left panel) and MoNA TDCs (right panel).

2.1 Sweeper Magnet

The dipole sweeper magnet has a bending angle of 43° and a vertical gap of 14 cm. During the experiment it was set to a magnetic rigidity of 2.994 Tm, and a tungsten beam blocker was placed on the low rigidity side of the magnet in order to block unreacted beam particles. The focal plane box located after the magnet contains detectors which can be used to measure position, timing, and energy information of the charged fragments. The present experiment utilized a pair of 30 mm × 30 mm cathode readout drift chambers (CRDCs) located 182 cm apart to measure focal plane position and angle; a 65 cm long ion chamber to measure energy loss; and a thin (4.5 mm) plastic scintillator to measure time of flight (ToF).

The TDCs for each of the timing detectors were calibrated using a pulser set to a 20 ns period; an example calibration spectrum can be seen in the left panel of Figure 2.3. Time offsets were set based on target-out runs with the sweeper magnet set to beam rigidity. Running at this setting ensures that the central velocity of ^{48}Ca events in the focal plane is at 10.5 cm/ns, and dividing this into the known path length of 680 cm gives a travel time of 64.8 ns.

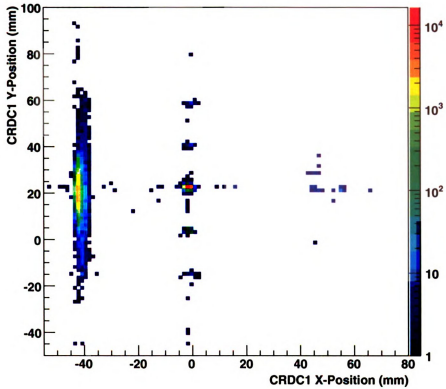


Figure 2.4: CRDC 1 y-vs-x spectrum with a mask placed in front of the detector.

CRDC x (dispersive) positions were calculated based on the deposited charge on each pad. The centroid of the charge distribution in pad space is calculated by first finding the pad with maximum deposited charge and then performing a gravity fit over all of the pads. A gaussian fitting routine is then called, using the centroid and width of the gravity fit as initial parameters. The resulting centroid of the gaussian fit is then defined as the x position in pad space. The slope needed to convert pad position to position in mm is simply the width of each pad: 2.54 mm. Offsets are determined using mask runs as described below.

Position in the y (non-dispersive) direction is measured based on the travel time of the avalanche charge from the interaction point to the pads. The slope and offset needed to turn the raw time signal into a calibrated position in mm is determined by placing a mask with holes and a slit drilled at known positions in front of the CRDC

detector. The beam is then swept across the focal plane to illuminate the entire mask. The measured CRDC position spectrum, as shown in Figure 2.4, only contain data at the locations of the mask holes and slit. The slope is calculated based on the spacing between the holes, while offsets in both the x and y directions are determined from the known position of the center hole. The L-shaped pattern seen in the right side of the figure is used to identify the center hole.

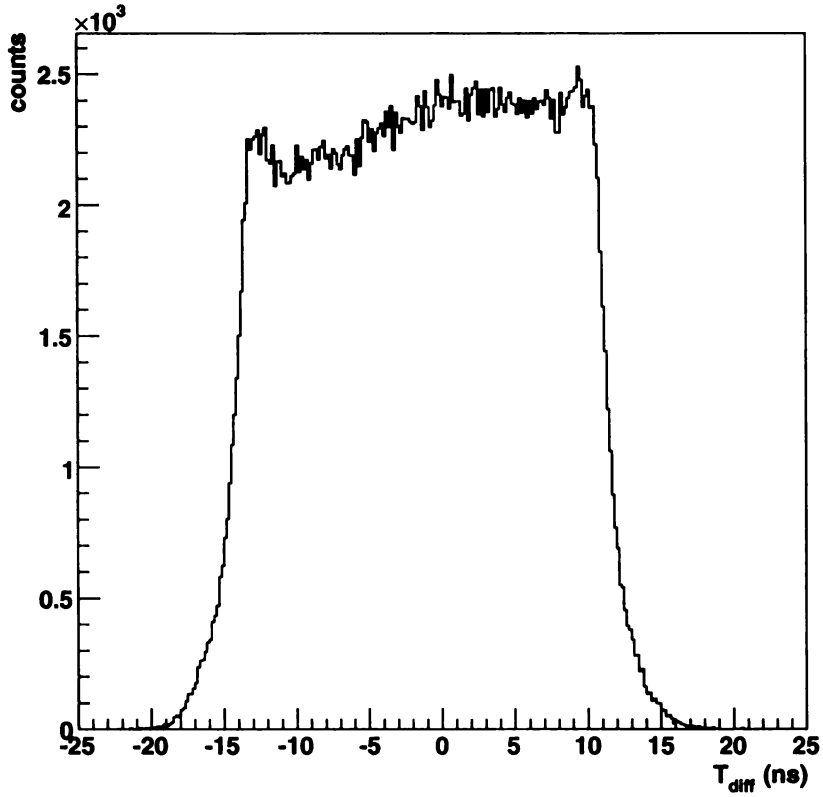


Figure 2.5: Time difference spectrum for the center bar in MoNA's front layer.

2.2 MoNA

MoNA is an array of 144 plastic scintillator modules, each 10 cm high x 200 cm wide x 10 cm deep. The scintillation light is detected on both ends of the modules by photomultipiler tubes (PMTs). ToF is determined by the mean of the timing signals of the two PMTs, while horizontal position is determined by the time difference between the two PMT signals, as explained below. Vertical and lateral position are determined based on the module in which the neutron interacts.

In order to convert the time difference between the left and right PMTs into a calibrated position, a simple linear conversion was used:

$$x = m \cdot t + b, \tag{2.1}$$

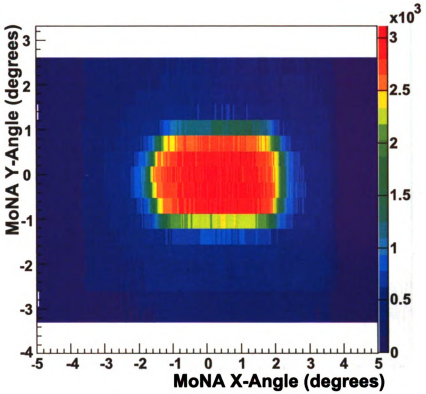


Figure 2.6: Plot of MoNA y-angle vs. x-angle, demonstrating the limited acceptance caused by the quadrupole tiplet.

where x is the position in the bar, t is the time difference between the two PMTs, and m and b are the linear slope and offset, respectively. The needed parameters m and b are determined experimentally using cosmic ray muons; Figure 2.5 shows an uncalibrated time difference spectrum for one MoNA bar, taken from a cosmic ray run. The physical edges of the bar are defined to be at 1/3 of the maximum height on each edge of the spectrum, based on Monte Carlo simulations. Using the 200 cm length of the bar, along with the position in time space of each edge, m can be calculated to be:

$$m = \frac{200 \text{ cm}}{t_{\text{right}} - t_{\text{left}}}, \tag{2.2}$$

where t_{right} and t_{left} are the right and left edges of the bar in time space. The

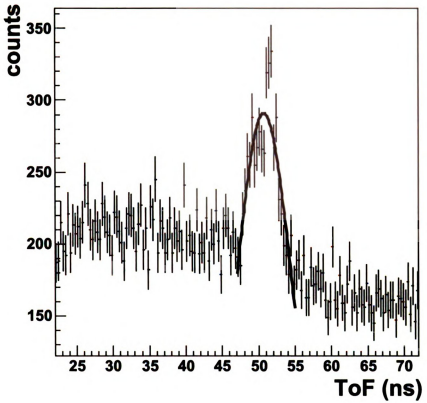


Figure 2.7: ToF spectrum for γ-rays from the production target to the center of the front layer of MoNA.

offset, b, can then be determined by setting $x = 0$ at the halfway point between t_{left} and t_{right}, $(t_{\text{left}} + t_{\text{right}})/2$, and solving for b:

$$b = (t_{\text{right}} + t_{\text{left}}) \cdot \frac{m}{2}. \tag{2.3}$$

In the present experiment, MoNA was arranged in a configuration 16 modules high by 9 modules deep, and the front face was placed at a distance of 15.38 m from the reaction target. In this configuration, neutrons are shadowed by the opening bore of the triplet, resulting in an angular acceptance of $\pm 1.5°$ in the vertical direction and $^{+2.5°}_{-2.0°}$ in the horizontal direction, as shown in Figure 2.6.

MoNA TDCs were calibrated using a pulser set to a range of 320 ns and period of 40 ns. The full range of each TDC was set to 350 ns. The time calibrator sends a

pulsed signal every 40 ns that is read into each MoNA TDC spectrum, as shown in in Figure 2.3. The slope needed to convert raw channels into nanoseconds is determined by simply dividing the average spacing between the spikes shown in the figure into 40 ns.

The overall time offset for MoNA was calculated using γ-rays originating from the production target. Using the known γ-ray velocity of 29.998 cm/ns and the measured distance from the production target to the center of the front layer of MoNA of 15.38 m, the ToF of a γ-ray from the target to the center of the front layer of MoNA was calculated to be 51.3 ns. The TDC offset was then set such that the central γ-ray ToF falls at 51.3 ns, as shown in Figure 2.7.

Chapter 3

Data Analysis

3.1 Particle Identification

The charged fragments are separated and identified using energy loss and ToF measurements. The element (Z) identification comes primarily from the energy loss signal in the ion chamber, while mass number (A) is determined from ToF. As can be seen in Figure 3.1, the uncorrected ToF signal does not provide sufficient resolution to separate the fragments, due to the large momentum acceptance of the sweeper magnet. Hence the ToF must be corrected event by event, based on the magnetic rigidity of the fragment.

Since position and angle in the dispersive (x) plane correlate with magnetic rigidity, the needed correction is performed based on x and θ_x at the focus of the triplet-sweeper magnet setup. The method used to correct the ToF is to apply linear correction factors, as demonstrated in the following equation:

$$t_{corr} = t + c_x \cdot x + c_\theta \cdot \theta, \tag{3.1}$$

where c_x and c_θ are empirically determined constants. Due to nonuniformities of the magnetic field, the correction factors depend on the location of the particle in angle-position phase space. To account for this effect, the phase space is divided into

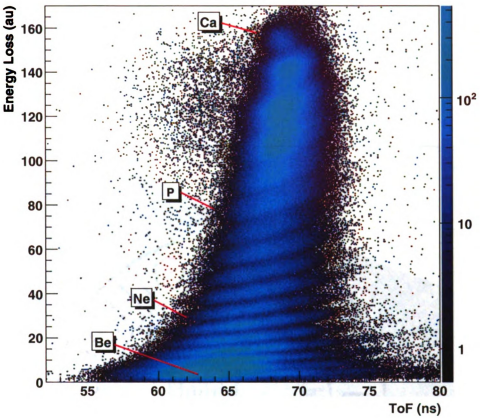


Figure 3.1: Uncorrected energy loss vs. ToF plot, with selected elements indicated.

a 5×6 grid, as shown in Figure 3.2. Separate correction factors are then determined for each grid region. As can be seen in Figures 3.3 and 3.4, the correction factors c_x and c_θ trend upwards as x and θ increase, respectively. Figure 3.5 shows a sample particle ID plot for one grid region, using corrected ToF.

Neutrons are identified simply by their ToF. Charged particles are deflected by the sweeper, and the neutrons are cleanly separated from prompt γ-rays as shown in Figure 3.6. Background neutron events are subtracted as described in Section 4.2.

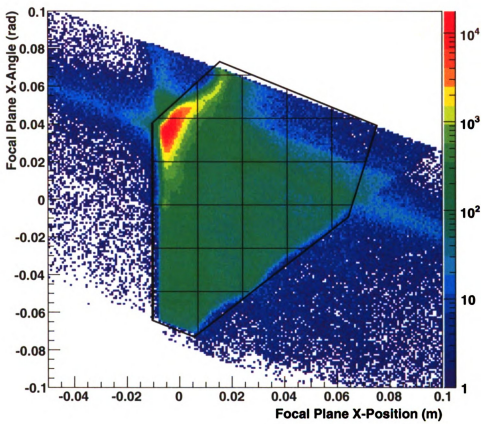


Figure 3.2: Focal plane angle vs. position. The black outline indicates events which are accepted for analysis, and the inner lines are the boundaries of the grids used for calculating ToF corrections. The peak around $\theta_x \simeq 0.04$ rad, $x \simeq 0.005$ m is composed primarily of scattered background events, which are excluded as described in section 3.2.

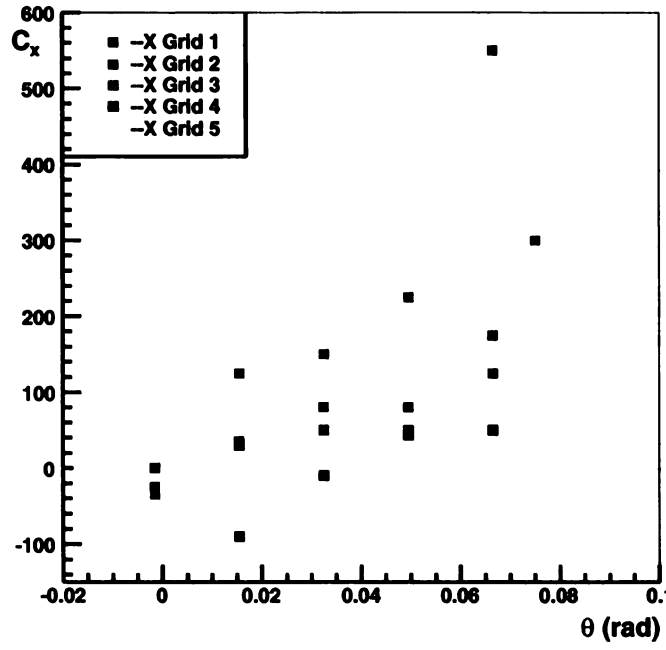


Figure 3.3: x-correction factor (c_x) as a function of θ position.

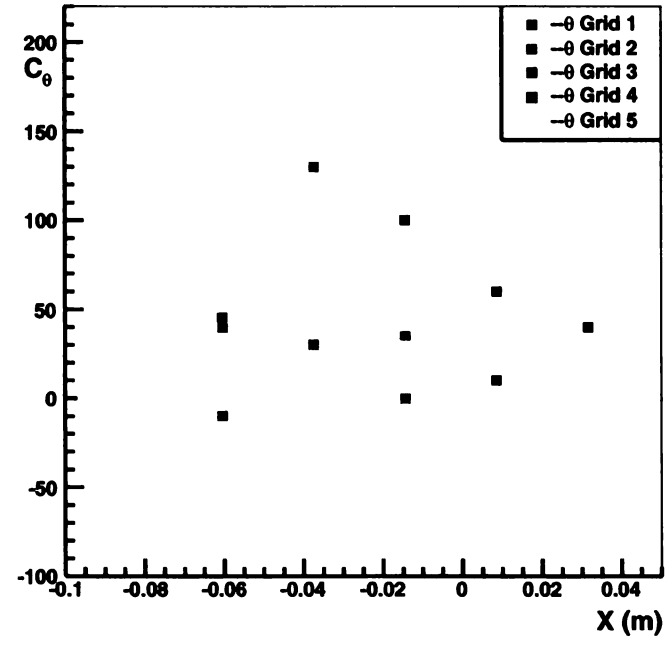


Figure 3.4: θ-correction factor (c_θ) as a function of x position.

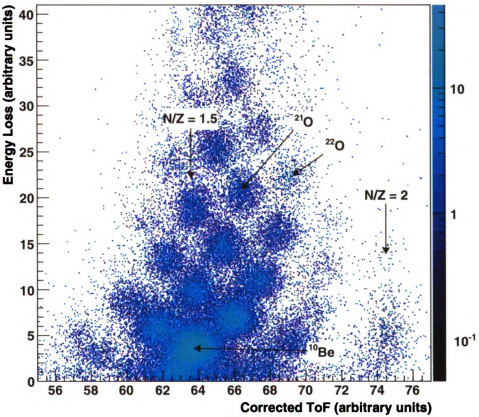


Figure 3.5: Sample particle identification plot. The vertical $N/Z = 2$ and $N/Z = 1.5$ bands, along with selected isotopes, are indicated in the figure.

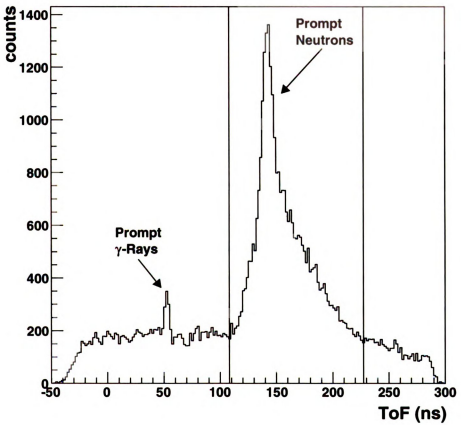


Figure 3.6: MoNA ToF spectrum, gated on ^{10}Be, showing clear separation between prompt neutrons and prompt γ-rays.

3.2 Scattered Background

The upper left panel in Figure 3.7 shows a double peak in velocity for ^{20}O. This double peak structure is present in nearly all of the isotopes observed, and comparison with coincident neutron spectra rules out the possibility of the double peak structure being a result of the kinematics of the neutron emission. Further insight into the cause of this double peak structure can be gained by plotting focal plane dispersive angle (θ_x) versus fragment velocity, as shown in the remaining three plots in Figure 3.7. Events falling outside of the outlined oval regions in these three plots are identified to be background events, as explained below.

The sweeper magnet curves the trajectory of particles with a lower magnetic rigidity more strongly than those with higher magnetic rigidity. A more drastic bending corresponds to a more negative value of θ_x. Since magnetic rigidity is proportional to mv/q, there is a positive correlation between θ_x and v_{frag}—slower particles are bent more, making θ_x more negative, while faster particles are bent less making θ_x more positive. This positive correlation between θ_x and v_{frag} can clearly be seen for the events falling inside the ovals in Figure 3.7. Events falling outside of the ovals have the opposite correlation—slower fragments come with a more positive angle. This correlation indicates that these events must not have taken an unobstructed path through the sweeper magnet.

The events shown in Figure 3.7 which fall outside of the oval regions do come in coincidence with neutrons. These neutrons have a normal ToF and position distribution, indicating that they were produced at the reaction target. This makes it unlikely that the suspected bad events in Figure 3.7 are due to scattered primary beam and instead indicates that they are due to scattered reaction products.

Tracing the path of the suspected background particles through the sweeper's magnetic field provides further evidence that these events must be the result of scattering. The nominal $B\rho$ value of the suspected background events is approximately

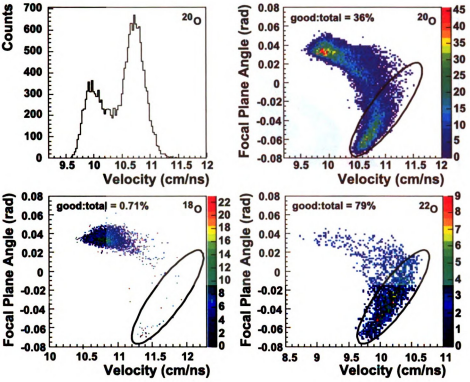


Figure 3.7: Focal plane angle versus velocity plots for selected Oxygen isotopes. The plot in the upper left corner is a projection onto the velocity axis for ^{20}O. Events falling outside of the indicated ovals are attributed to scattering off the low rigidity side of the sweeper magnet.

2.8 Tm, based on the velocity of coincident neutrons. Tracking of a 2.8 Tm particle through the field of the sweeper magnet results in the particle exiting the magnet in an area covered by the tungsten blocker, as shown in Figure 3.8. Clearly the particles in question can not have arrived in the focal plane by taking an unobstructed track through the magnet.

Based on the arguments presented above, events falling outside of the region in which θ_x and v_{frag} are positively correlated are excluded from the analysis.

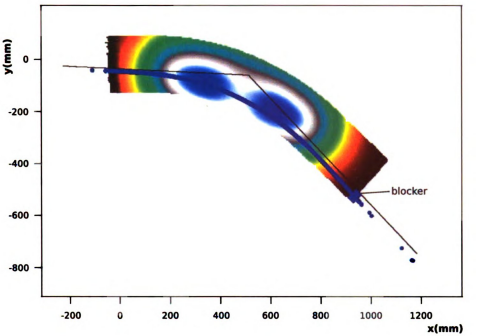


Figure 3.8: Track of a 2.8 Tm particle through the magnetic field of the sweeper. The blue curve is the path of the 2.8 Tm particle, while the black line represents the central trajectory. The location of the tungsten blocker is indicated on the figure.

3.3 Decay Energy Reconstruction

Invariant mass spectroscopy is used to reconstruct E_{decay}, according to Equation 1.3, which is reproduced here:

$$E_{\mathrm{decay}} = \sqrt{m_{\mathrm{f}}^2 + m_{\mathrm{n}}^2 + 2\left(E_{\mathrm{f}} \cdot E_{\mathrm{n}} - p_{\mathrm{f}} \cdot p_{\mathrm{n}} \cdot \cos\left(\theta_{\mathrm{open}}\right)\right)} - m_{\mathrm{f}} - m_{\mathrm{n}}.$$

The equation contains the masses, energies and momenta of the neutrons (m_{n}, E_{n}, p_{n}) and the fragments (m_f, E_{f}, p_{f}) as well as the opening angle θ_{open} between the two particles.

The neutron angle is calculated from the measured interaction point in MoNA, and the energy and momentum is calculated using ToF and flight distance. Likewise for the charged fragment, the energy is calculated using ToF and path length, while the angle is reconstructed from a partial-inverse matrix produced using COSY INFINITY [26].

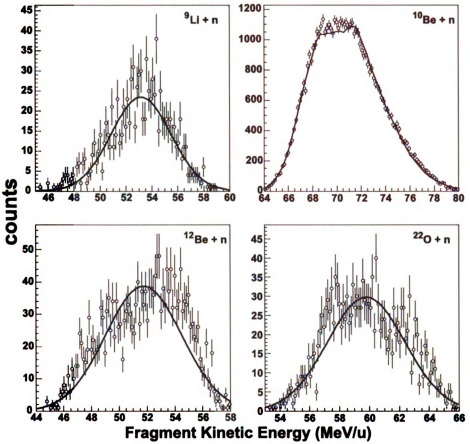


Figure 3.9: Fragment energy distributions at the reaction point for ^{9}Li ^{10}Be, ^{12}Be and ^{22}O. Open circles are the data and solid lines simulation.

3.4 Monte Carlo Simulations

In order to account for the experimental resolution and acceptance effects that are present in the data, Monte Carlo simulations are performed which fold these properties into a theoretical lineshape. The code used presently is an extension of a code originally written by H. Scheit for the analysis of ^{23}O and ^{24}O neutron decays which were studied in the Sweeper-MoNA setup [8,27,28]. The simulation begins by describing the position, angle and energy of the incoming ^{48}Ca beam, using a GSL gaussian random number generator [29]. It then strips the beam of the number of protons and neutrons needed to go from ^{48}Ca to the excited fragment of interest.

An attempt was made to approximate the kinematics of the ^{48}Ca fragmentation

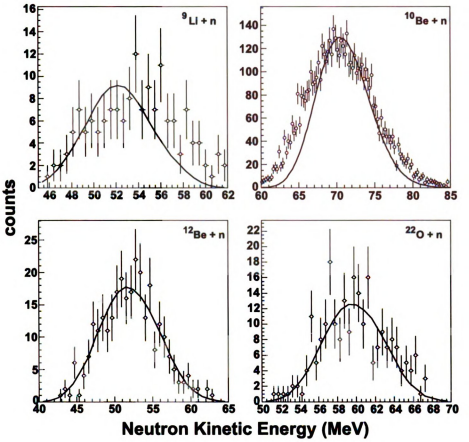


Figure 3.10: Kinetic energy distributions (open circles) and simulation results (solid lines) of neutrons in coincidence with ^{9}Li, ^{10}Be, ^{12}Be and ^{22}O. Both the data and the simulation are gated on events with $E_{\text{decay}} < 20$ keV.

reaction using a Glauber model [30]. However, due to the violent nature of the reaction and the limited $B\rho$ acceptance of the sweeper magnet, the shape of the energy distribution of fragments recorded into the data stream is dominated by acceptance effects rather than the shape of the kinetic energy distribution at the target. This results in many of the fragment energy distributions being peaked at values significantly higher or lower than the incoming beam energy, meaning that the recorded data are taken from a small slice of the tail of the energy distribution at the target. Since the precise shape of the energy distribution in this tail region is not known, and because accepting only a small fraction of the generated events causes the necessary

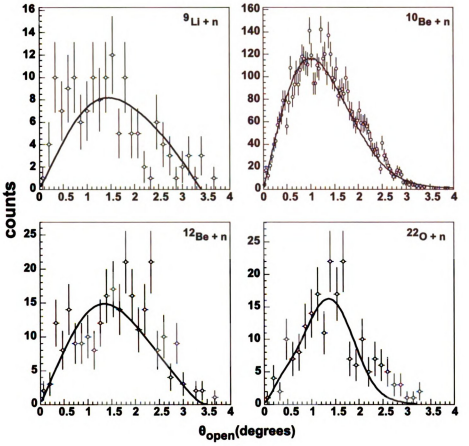


Figure 3.11: Opening angle distributions for ^{9}Li + n, ^{10}Be + n, ^{12}Be + n and ^{22}O + n, for events with $E_{decay} < 20$ keV. Open circles are experimental data and the solid lines are simulation results.

run time of the simulation to become prohibitively large, it was determined that it is not practical to simulate the fragment energy distributions using a kinematical model.

The method used to describe the fragment energy distribution is to simply give the incoming beam an energy distribution which matches a fit to the kinetic energy distribution of the data. Because of the fragment's heavier mass, calculation of decay energy is dominated by the kinematics of the neutron; hence the influence of using a fitted input fragment energy on the final decay energy curve is negligible. Measured kinetic energy distributions of the four fragments observed with a low energy resonance (^{9}Li, ^{10}Be, ^{12}Be and ^{22}O)[1] are shown in Figure 3.9 (open circles), along with

[1]It should be noted here that the labels ^{9}Li, ^{10}Be, ^{12}Be and ^{22}O here refer to the species of

fits to the data which are input into the simulation (solid lines). For ^{9}Li, ^{12}Be and ^{22}O, the shape of the energy distribution is approximated as a gaussian, while ^{10}Be is fit with a gaussian tail on either side of the curve and a flat distribution in the middle.

The next step in the simulation program is to model the de-excitation of the excited nucleus via neutron emission. In this step, the excited nucleus loses one neutron, resulting in a system composed of the neutron and the remaining charged fragment. The relative energy distribution of the neutron and charged fragment is generated based on what is known about the spectroscopic properties of the nucleus in question. The relative energy distributions used to model the decay of the present data are discussed in Section 4.2.

After it exits the target, the charged fragment is tracked to the focal plane of the sweeper magnet using a COSY forward map [26]. Here cuts are placed on the data such that only those events which fall inside the physical dimensions of the focal plane detectors are accepted. Software cuts equivalent to those used to filter out scattered background data are also applied at this time. In order to simulate the resolution of the CRDC and timing detectors, the position, angle and time values are given a gaussian spread, based on the known resolutions of the detectors. From these randomized focal plane positions and angles, the angle of the fragment at the reaction point is reconstructed using a four parameter partial inverse matrix [31,32] produced using COSY. The energy of the fragment is also calculated based on the ToF and mean flight path.

The neutron is tracked to its position in the first layer of MoNA. At this point, its horizontal position is given a gaussian spread to simulate resolution, and its vertical position is set to the center of the bar in which it interacts. The lateral position of the interaction is inconsequential to the calculation of neutron energy; hence all neutrons are assumed to interact in the first layer of MoNA. However, the lateral

charged fragment remaining after the excited nucleus decays via neutron emission.

position within this layer is randomized to reproduce the effects of ToF resolution. Neutron energy is then calculated based on the randomized ToF and flight distance, and the angle of the neutron as it exits the target is calculated from the randomized interaction point in MoNA.

From the neutron and fragment energies and opening angle, a simulated decay energy distribution is calculated, using Equation 1.3. Since the parameters that go into calculating this decay energy include experimental resolution and acceptance effects the resulting curve can be compared directly to data. Plots of the simulated decay energy curves for the fragments ^{9}Li, ^{10}Be, ^{12}Be and ^{22}O are presented along with a discussion of the experimental results in Section 4.2.

As can be seen in Equation 1.3, the parameters which are most important to calculation of the decay energy are fragment kinetic energy, neuron kinetic energy and the opening angle between the fragment and the neutron. For the fragments ^{9}Li, ^{10}Be, ^{12}Be and ^{22}O, Figures 3.10 and 3.11 show comparisons between simulation (solid lines) and data (open circles) for kinetic energy and opening angle, respectively. The figures exclude events for which $E_{\mathrm{decay}} \geq 20$ keV. This is done to reduce the effect of background neutron events which dominate at higher decay energies, as demonstrated in Section 4.2.

Detailed instructions on how to run the simulation program can be found in Appendix A.

Chapter 4

Results and Discussion

4.1 Results

The isotopes which could be identified and separated and which were produced in sufficient quantities to be studied are indicated on a chart of nuclides in Figure 4.1. The range of isotopes is determined by the $B\rho$ and the momentum acceptance of the sweeper magnet. For $Z > 10$ the ToF resolution is not sufficient to separate neighboring isotopes.

Figure 4.2 shows relative velocity spectra for the three most prevalent fragments of each element from lithium through neon. The spectra of ^{9}Li, ^{10}Be, ^{12}Be, and ^{22}O show a peak near zero relative velocity, suggesting the presence of a neutron-unbound state with small decay energy. The enhancement near zero relative velocity in ^{11}Be is attributed to contamination from ^{10}Be. All other fragments, including those not displayed in the figure, have a rather broad shape indicative of a background distribution.

The presence of a coherent neutron decay in ^{9}Li, ^{10}Be, ^{12}Be, and ^{22}O can be confirmed by doing an event mixing calculation to give an estimate of the background arising from uncorrelated neutron events. For a given fragment, the event mixing is performed by pairing each fragment event with a neutron that comes in coincidence

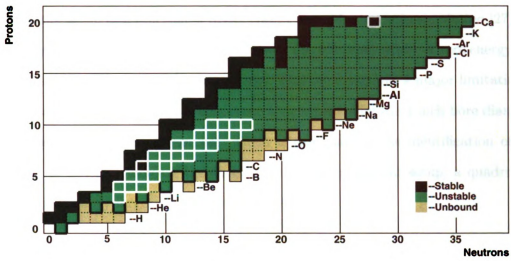


Figure 4.1: Chart of the nuclides. Isotopes outlined in white were observed in sufficient quantities to be studied.

with a different isotope. The decay energy calculation of Equation 1.3 is then carried out using the mixed events. Figure 4.4 shows the results of this event mixing calculation, superimposed with the experimental data, for ^{9}Li, ^{10}Be, ^{12}Be, and ^{22}O. For each of the four fragments, there is clearly a low energy peak rising above the event mixed spectrum. In contrast, Figure 4.5 shows event mixed and real data histograms for four representative fragments that do not have a low energy enhancement: ^{12}B, ^{18}N, ^{22}F, and ^{26}Ne. In this figure, the real data and event mixed histograms for these fragments show no discernable difference.

As can be seen from Figures 4.2 and 4.3, the majority of the isotopes produced in this experiment show no evidence of a coherent neutron decay. These isotopes possibly possess a continuum of states above the one neutron separation energy, and since the experimental resolution of the present setup is not sufficient to separate closely spaced states, the resulting spectrum resembles a broad background distribution. Another possibility is that there is an isolated unbound state present, but that it is at too high of an energy to be observed, c.f. Figure 4.6.

The observation of only low decay energy states is not surprising and is due to the

limited angular acceptance of the experimental setup. Figure 4.6 shows the efficiency of the entire setup as a function of decay energy for a Monte-Carlo simulation [27] of a flat decay energy distribution from 0 to 1 MeV. The efficiency peaks at zero energy and then falls off rapidly with increasing energy. This demonstrates a major limitation of the method. The small angular acceptance is determined by the 4 inch bore diameter of the focusing quadrupole magnet, which is essential for the identification of the isotopes. In order to increase the acceptance using the present setup, a quadrupole with a significantly larger bore would be necessary.

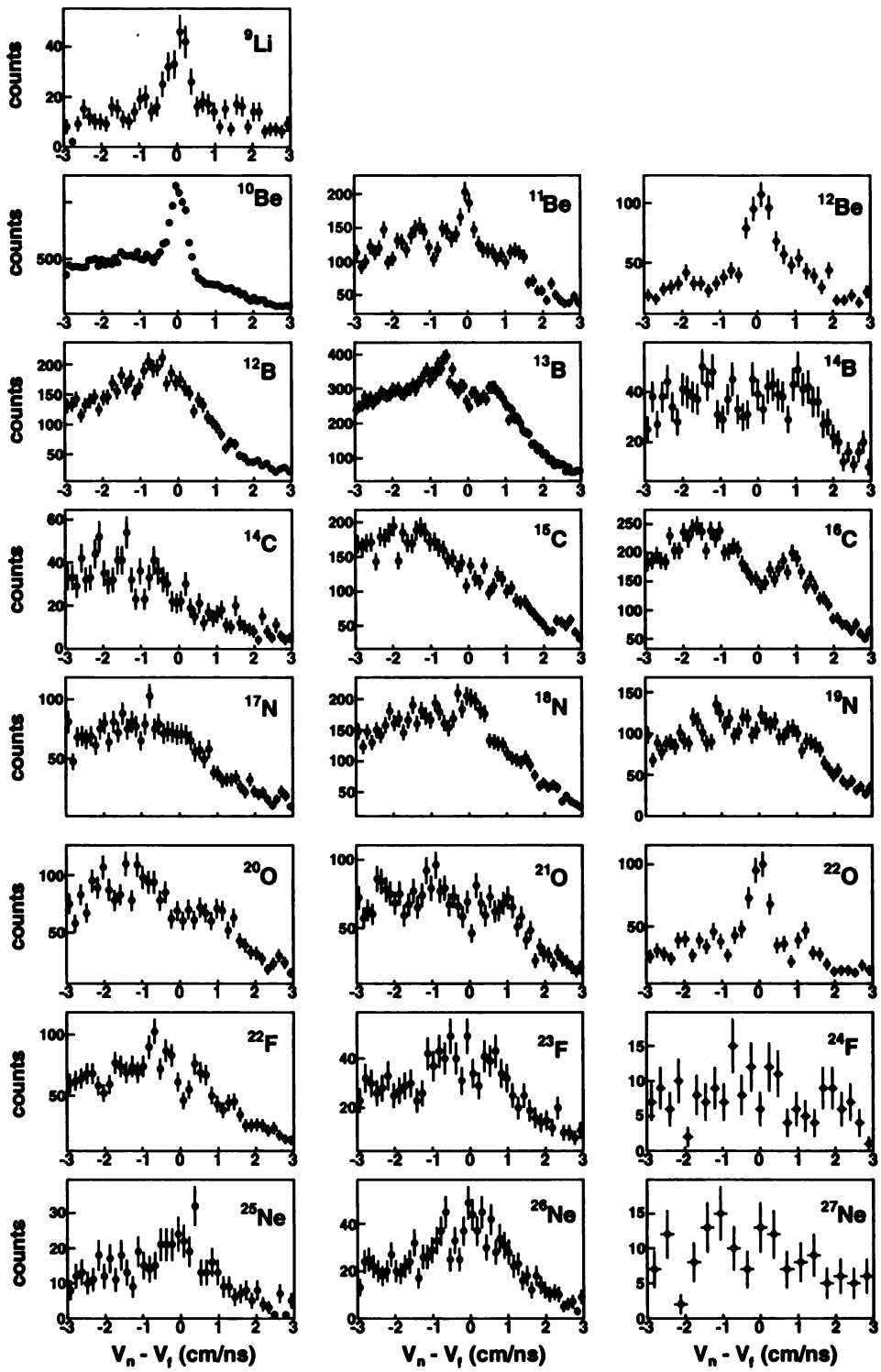


Figure 4.2: Velocity difference (neutron minus fragment) for the most prevalent isotopes.

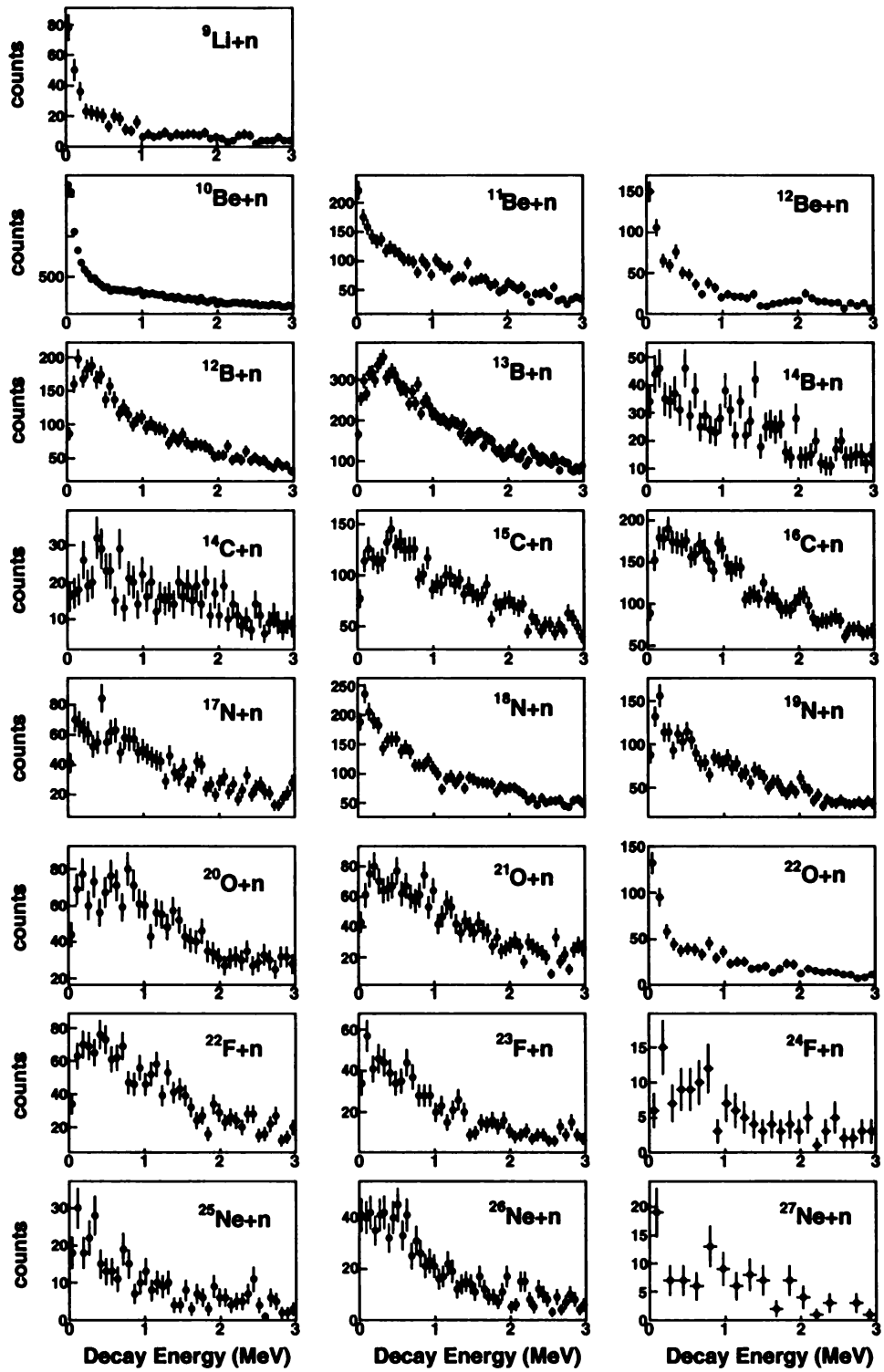


Figure 4.3: Decay energy spectra for the most prevalent isotopes.

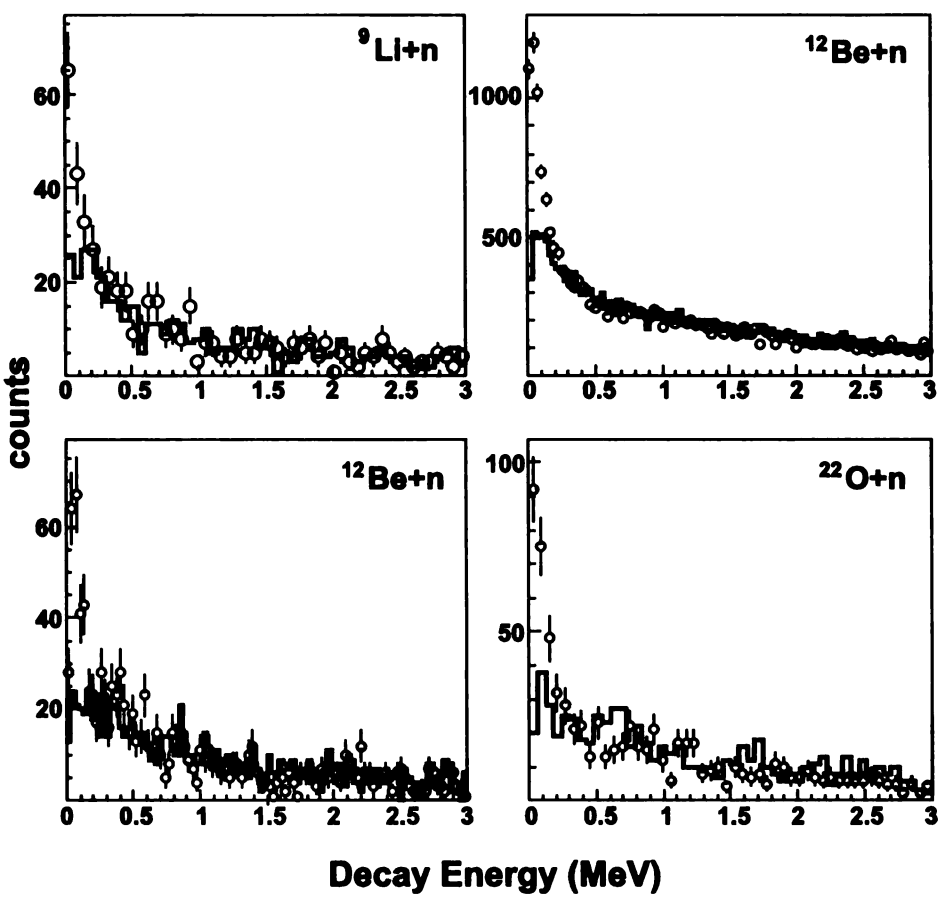


Figure 4.4: Decay energy spectra (open circles) for isotopes showing evidence of a resonance, with event mixing background (blue histograms) superimposed.

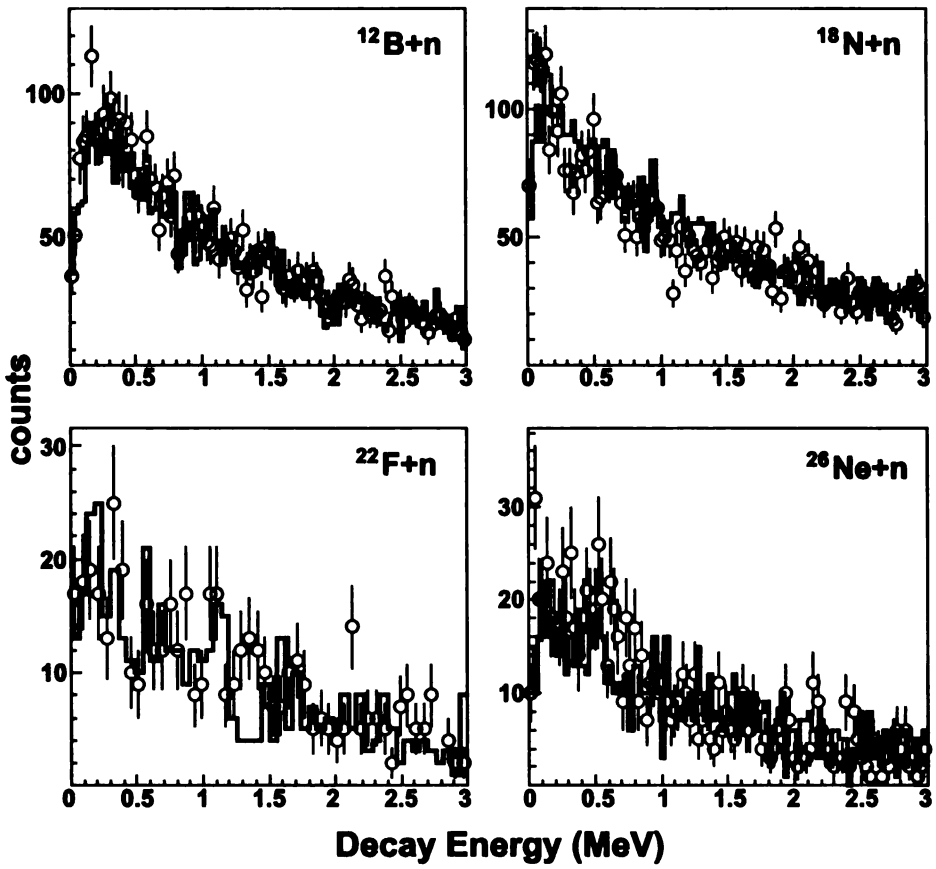


Figure 4.5: Decay energy spectra (open circles) for isotopes without a resonance, with event mixing background (blue histograms) superimposed.

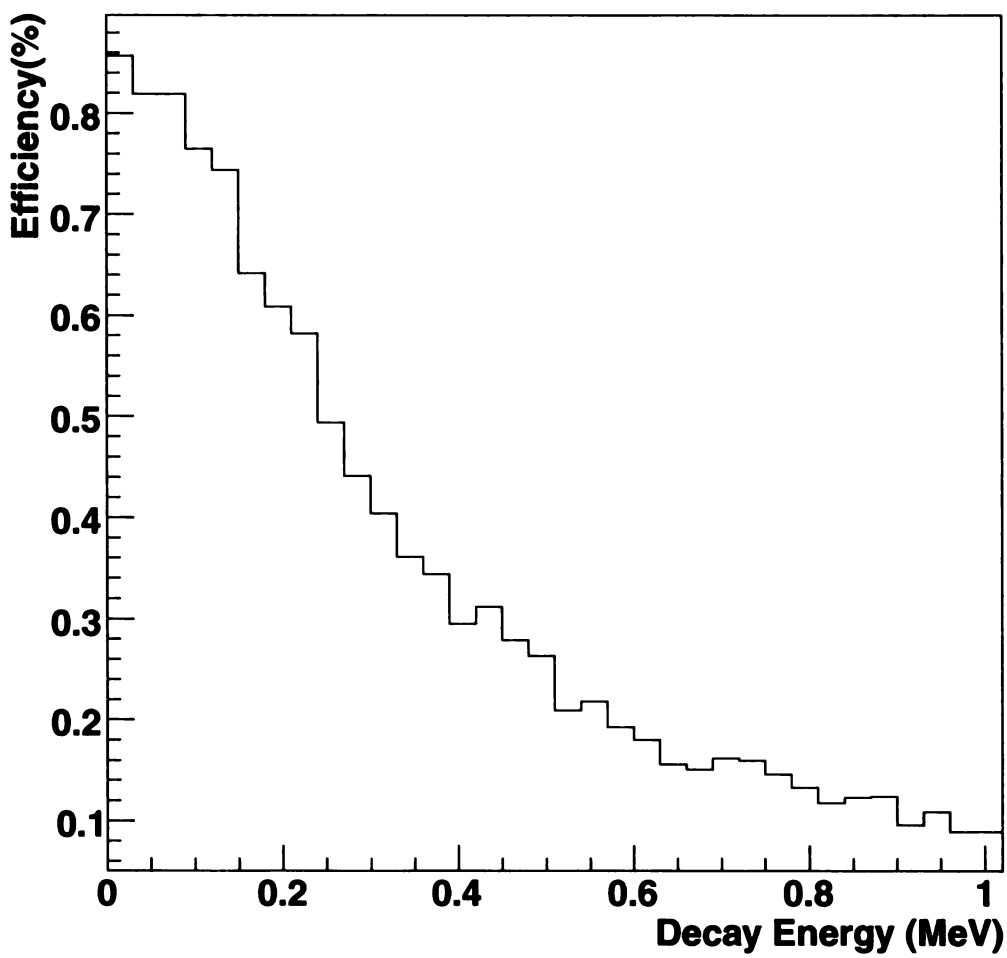


Figure 4.6: Efficiency the experimental setup as a function of decay energy.

4.2 Analysis of Low Energy Decays

In the following, the spectra of the four isotopes showing evidence for the presence of a low energy neutron decay state are analyzed in more detail. Experimental results from the literature are presented for each nucleus, along with a brief overview of theoretical considerations. The present data are then compared with past results. The intent of this analysis is not to make independent measurements of the decay properties of these nuclei, but rather to confirm that the present data are consistent with previous measurements.

For each of the four isotopes, Monte Carlo simulations were performed as described in Section 3.4, with the decay parameters for each individual isotope taken from the literature. The results of these simulations and experimental data for the fragments ^{9}Li, ^{10}Be, ^{12}Be, and ^{22}O, which originate from states in ^{10}Li, ^{11}Be, ^{13}Be and ^{23}O, respectively are shown in Figures 4.8, 4.12, 4.15 and 4.18. In the figures, background contributions calculated by event mixing, as shown in Figure 4.4, are subtracted from the data.

^{10}Li

The ground state of the neutron unbound nucleus ^{10}Li has been well studied [33], as knowledge of the ground state properties of ^{10}Li is essential for an understanding of the Borromean nucleus ^{11}Li [19, 34]. The general consensus is that the ground state of ^{10}Li is a virtual s-wave resonance in the ^{9}Li + n system [33, 34]. Experiments measuring ground state properties of ^{10}Li are analyzed in terms of the scattering length a_s, which is related to decay energy via:

$$a_s = -\hbar/\sqrt{2\mu E}, \tag{4.1}$$

where μ is the reduced mass of the ^{9}Li + n system. A summary of previous results is presented in Table 4.2. The most recent measurements [34] extract $a_s = -30^{+12}_{-31}$ fm,

Year	Authors	Reaction	$-a_s$ (fm)	Reference
1990	A.L. Amelin *et al.*	^{11}Be(π^-,p) ^{11}Li	$30^{+\infty}_{-20}$	[35]
1993	R.A. Kryger *et al.*	^{nat}C(^{18} O, ^{9}Li+n)X	≥ 10	[18]
1994	B.M. Young *et al.*	^{11}B(^{7}Li, ^{8}B) ^{10}Li	≥ 15	[36]
1995	H. Zinser *et al.*	^{nat}C(^{11}Be, ^{9}Li+n)X	≥ 20	[37]
1998	M.G. Gornov *et al.*	^{11}B(π^-,p)^{10}Li	$15^{+\infty}_{-5}$	[38]
1999	M. Thoennessen *et al.*	^{9}Be(^{18}O, ^{9}Li+n)X	≥ 20	[19]
2001	L. Chen *et. al*	^{9}Be(^{11}Be, X) ^{10}Li	≥ 20	[39]
2001	M. Chartier *et al*	^{9}Be(^{11}Be, X) ^{10}Li	≥ 20	[40]
2006	H.B. Jeppesen *et al.*	^{9} Li(^{2}H,p) ^{10} Li	13–24	[41]
2007	H. Simon *et al.*	^{nat}C(^{11}Li, ^{9}Li+n)X	30^{+31}_{-12}	[34]

Table 4.1: Summary of previous measurements of the scattering length of the ground state of ^{10}Li.

and the relative energy spectrum from this measurement is shown in Figure 4.7.

In the present work, the decay energy of ^{10}Li into ^{9}Li + n is simulated assuming an s-wave interaction, using the model described in Ref. [19]. A fit to the data at $a_s = -50$ fm, consistent with Ref. [34], is shown in Figure 4.8. This decay is shown on a level scheme in Figure 4.9.

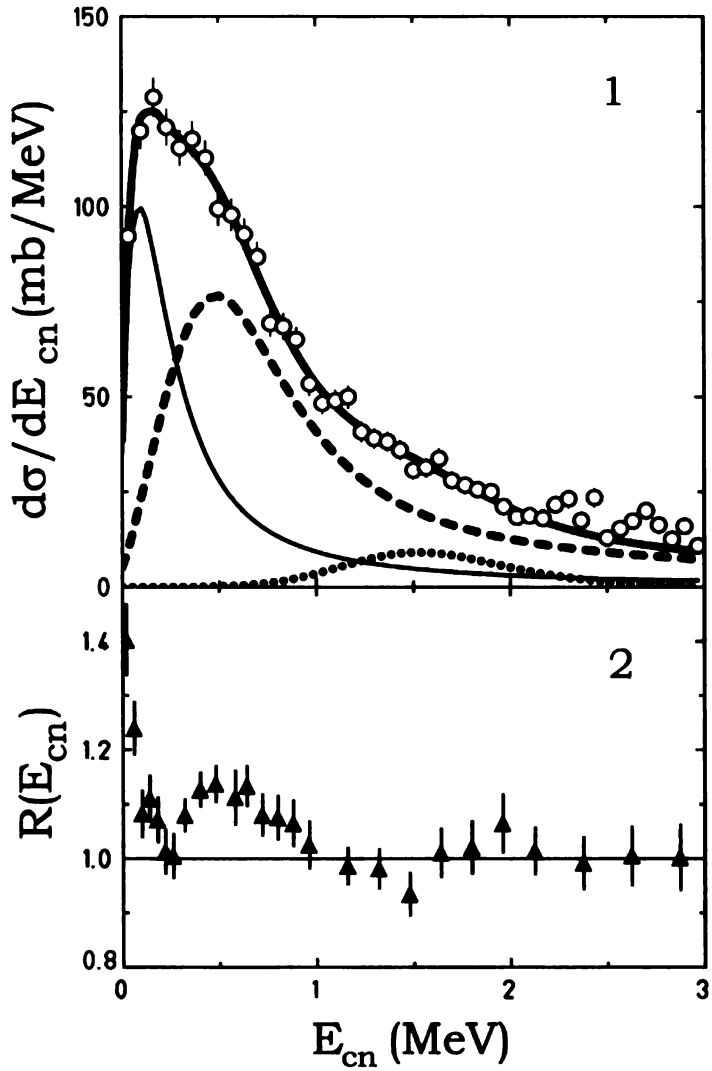


Figure 4.7: Relative energy spectrum (top panel) for ^{10}Li to ^{9}Li + n from Ref [34].

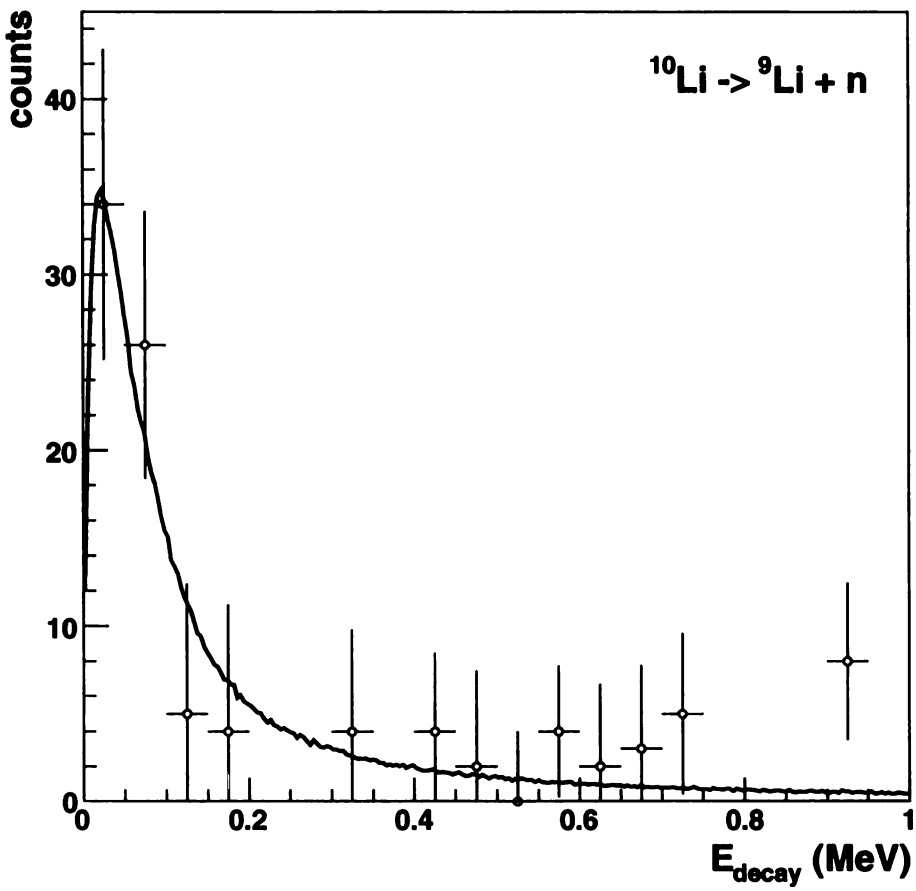


Figure 4.8: Background subtracted decay energy spectrum (open circles), with results from Monte-Carlo simulation superimposed.

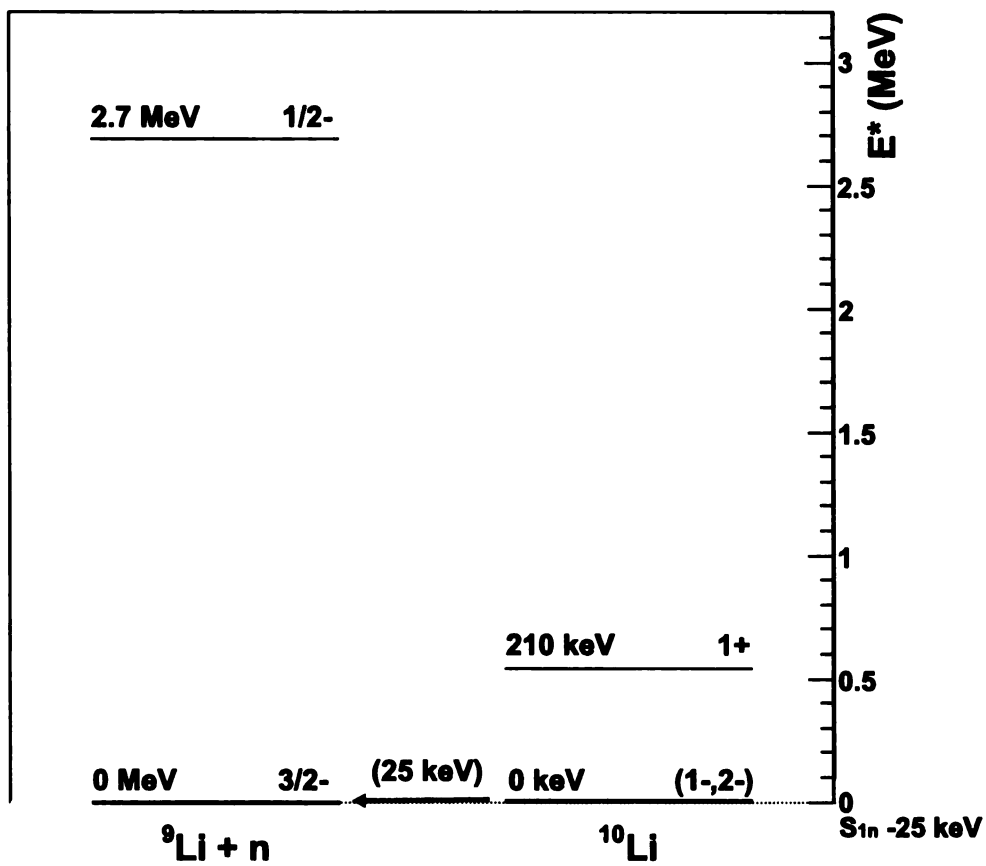


Figure 4.9: Level scheme showing the measured decay from ^{10}Li to ^{9}Li + n. Levels are taken from Ref. [42].

^{11}Be

^{11}Be is particle bound, with a one-neutron separation energy of 504 keV [42]. The lowest lying unbound excited state is located at 1778 keV [43]. A low energy neutron decay originating from ^{11}Be has been observed previously and attributed to a decay to the first excited 2^+ state at 3368 keV in ^{10}Be [13, 15, 44]. Deák *et al.* [15] quote a decay energy of 19(15) keV and assign this decay to the 3887(15) keV state in ^{11}Be [43]. Peters [44] states a decay energy of 84(15) originating from the 3956(15) keV $3/2^-$ state in ^{11}Be [43]. Pain *et al.* [13] do not attempt to assign the decay to a specific state in ^{11}Be and remark that the decay originates from excited ^{11}Be at approximately 4.0 MeV.

One-neutron knockout reactions selectively only populate specific states; therefore it is possible that the low energy state at 19 keV of Ref. [15] was not observed by Refs. [13, 44]. The relative velocity spectrum of Ref. [15] on the other hand was probably not sensitive enough to resolve a possible contribution of the state at 84 keV observed in Refs. [13,44]. Thus, the present decay of ^{11}Be to ^{10}Be + n is simulated with two Breit-Wigner resonances as described in [8]. The energies and widths of the two decays are $E_{\text{decay}} = 19$ keV; $\Gamma_{\text{decay}} = 10$ keV and $E_{\text{decay}} = 84$ keV; $\Gamma_{\text{decay}} = 10$ keV, based on the level assignments of Refs. [15] and [44], respectively. The widths for both decays are set to the upper limit of 10 keV, taken from Ref. [42]. Relative contributions from each of the two states are set as free parameters, and the best fit to the data is achieved with a 52.7% contribution from the 19 keV decay and a 47.3% contribution from the 84 keV decay. The fit is shown in Figure 4.12, along with the contributions from the 19 keV resonance (dashed) and the 84 keV resonance (dotted). A level scheme including the observed transitions is shown in Figure 4.13.

It should be mentioned that the data could also be described by a single resonance at $E_{\text{decay}} = 30$ keV, $\Gamma_{\text{decay}} = 65$ keV. This fit is shown as the blue line in figure 4.12. It is unlikely, however, that the data arise as the result of a single decay, due to the large width required in order to describe the data with a single resonance.

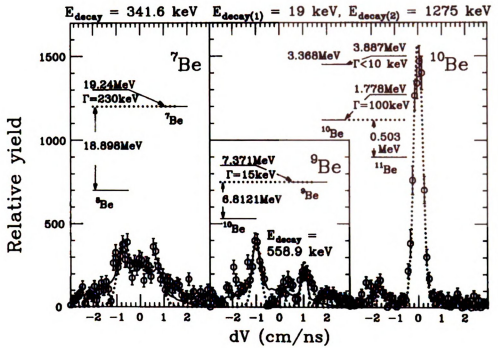


Figure 4.10: Relative velocity spectrum (rightmost panel) for ^{10}Be + n from Ref [15].

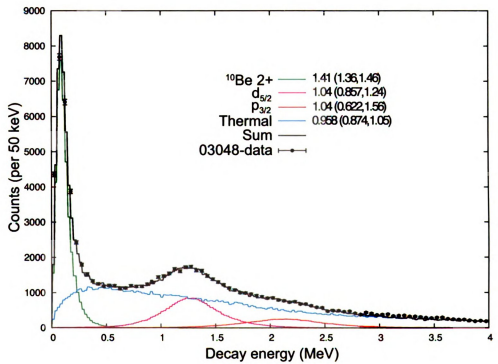


Figure 4.11: Decay energy spectrum for ^{11}Be to ^{10}Be + n from Ref [44].

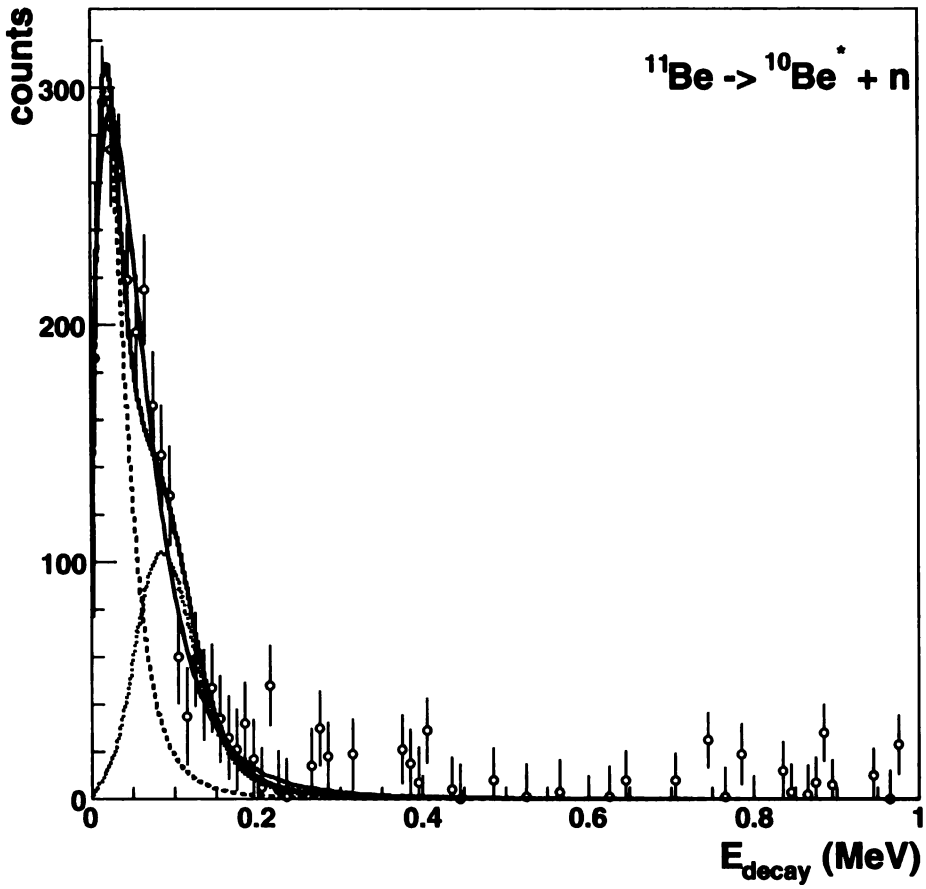


Figure 4.12: Background subtracted decay energy spectrum (open circles), with results from Monte-Carlo simulations (solid black line) for ^{10}Be + n. The Monte Carlo curve includes contributions from a 19 keV resonance (dashed line) and 84 keV resonance (dotted line). A simulated curve resulting from a single Breit-Wigner at $E = 30$ keV, $\Gamma = 65$ keV is also included (solid blue line).

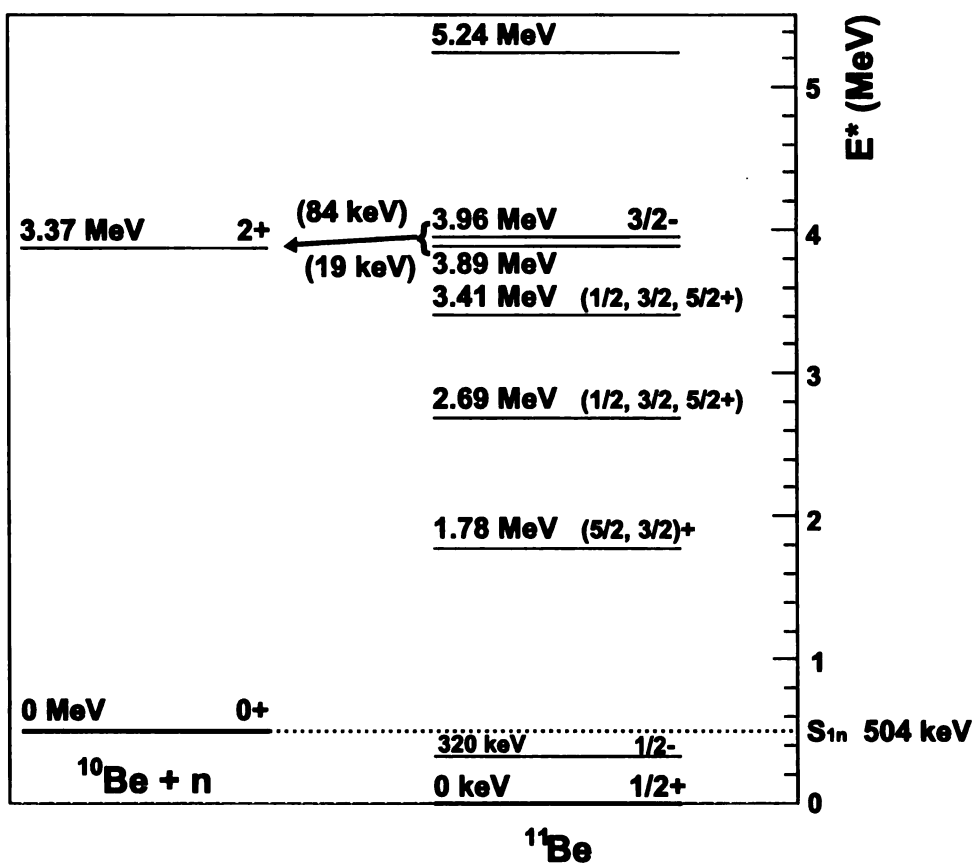


Figure 4.13: Level scheme showing the measured decay from ^{11}Be to ^{10}Be* + n. Levels taken from Ref. [42].

^{13}Be

Analogous the case of ^{10}Li, knowledge of the ground state properties of neutron unbound ^{13}Be is needed to understand the Borromean ^{14}Be nucleus [34]. Theoretical calculations predict the ground state of ^{13}Be to be an s-wave close to the neutron emission threshold [45, 46], though other models have suggested that it could be a p-wave at approximately 30 keV above threshold [47].

Experimental investigations into the nature of the ^{13}Be ground state have produced varied results. An SNDS experiment from a primary ^{18}O beam observed a low-lying s-wave resonance as the ground-state of ^{13}Be [20]. This study used relative velocity measurements to estimate the decay energy, and it extracted a scattering length of $a_s < -10$ fm. However, the possibility of a p or d state at $E_{\text{decay}} = 50(10)$ keV with a width of $\Gamma_{\text{decay}} \leq 10$ keV could not be ruled out. Subsequent observations using a single proton-knockout reaction from ^{14}B [48] or single neutron-knockout from ^{14}Be [34] failed to observe this low-lying resonance. Instead a strong resonance at approximately 700 keV was reported.

The present decay spectrum for ^{12}Be + n indicates a low-energy decay similar to Ref. [20]. Figure 4.15 shows the results of fits using an s-wave (solid line) and a Breit-Wigner resonance (dashed line). The parameters used for the fits are $a_s = -20$ fm (s-wave) and $E_{\text{decay}} = 60$ keV, $\Gamma_{\text{decay}} = 10$ keV (Breit-Wigner), based on the results presented in Ref. [20]. A level scheme placing the ground state of ^{13}Be slightly above ^{12}Be + n is also shown in Figure 4.16.

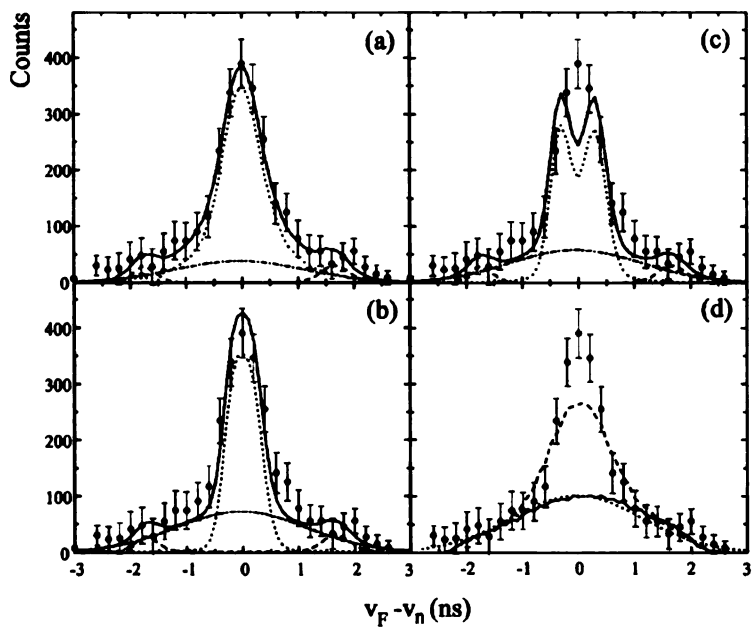


Figure 4.14: Relative velocity spectra for ^{12}Be + n from Ref [20].

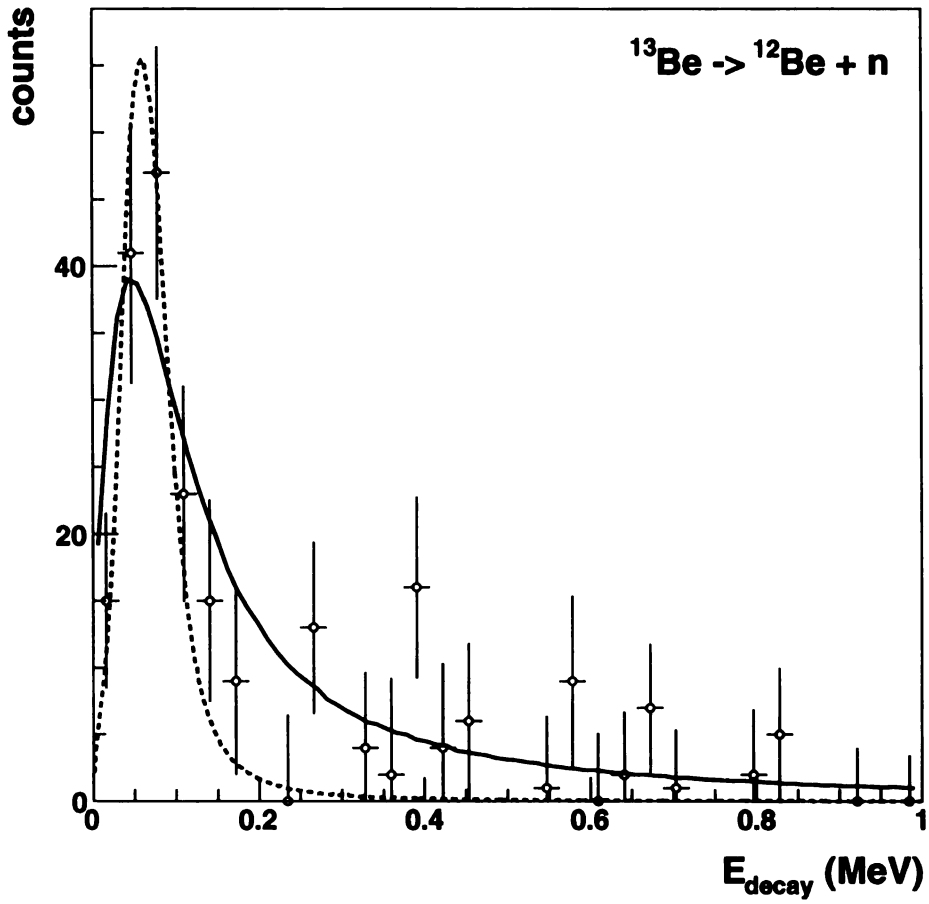


Figure 4.15: Background subtracted decay energy spectrum (open circles), with results from Monte-Carlo simulations superimposed. Two simulated curves are shown: an s-wave with $a_s = -20$ fm (solid line) and a Breit-Wigner with $E_{\text{decay}} = 60$ keV, $\Gamma_{\text{decay}} = 10$ keV (dashed line).

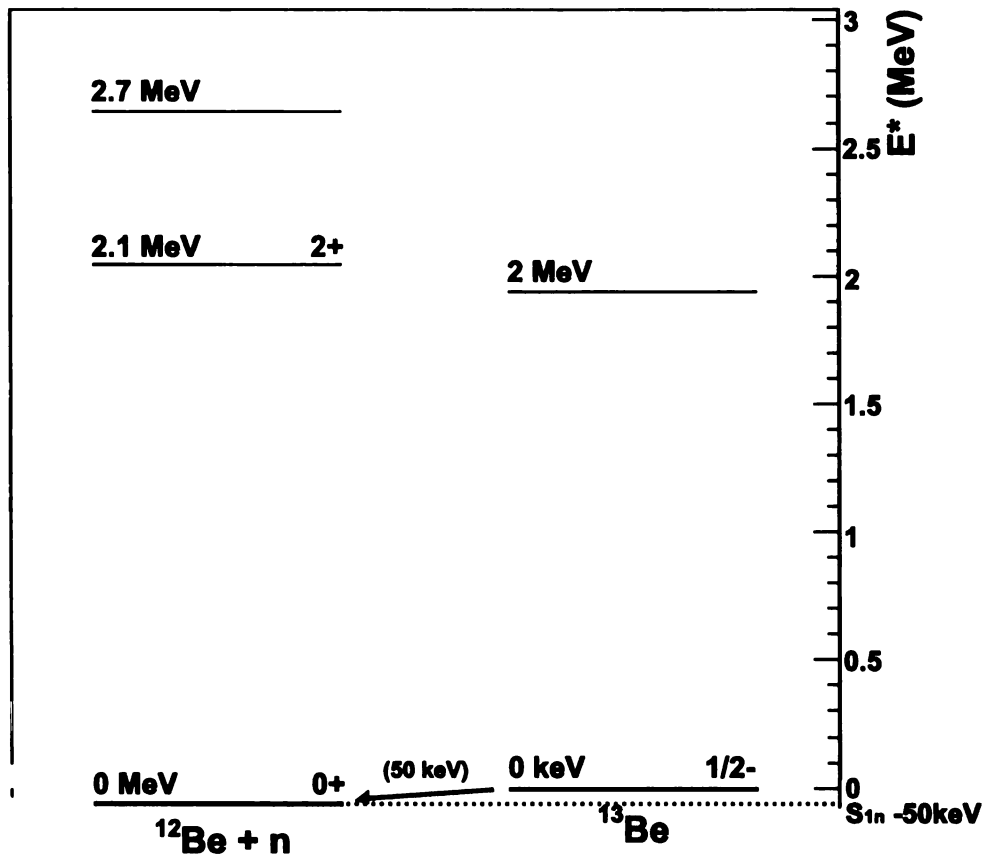


Figure 4.16: Level scheme showing the measured decay from ^{13}Be to ^{12}Be + n. The 2 MeV state in ^{13}Be is taken from Refs. [34, 48]; all other levels are from Ref. [42].

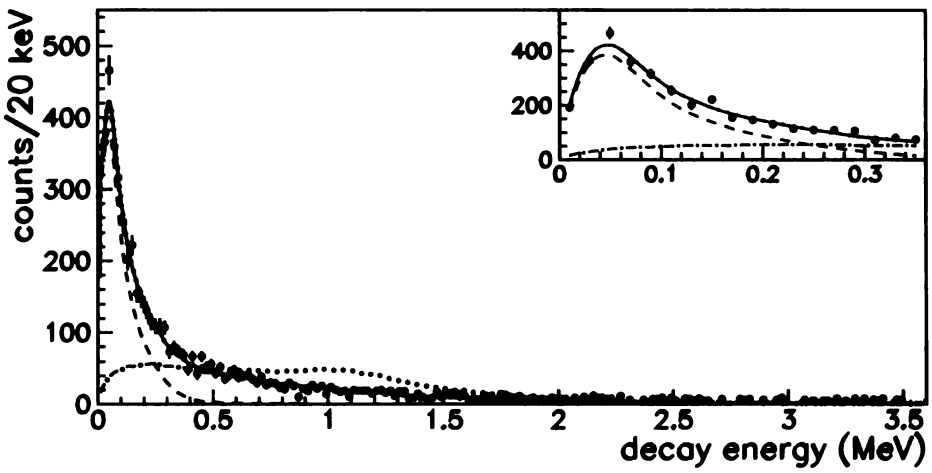


Figure 4.17: Decay energy spectrum for ^{23}O to ^{22}O + n published in Ref [8].

^{23}O

The Oxygen isotopes on either side of ^{23}O both show evidence of being magic nuclei, thus making knowledge of the excitation spectrum of ^{23}O important for shell model calculations [8]. An attempt to study the first excited state of ^{23}O using γ-ray spectroscopy [49] failed to observe a decay, indicating that the state is unbound with respect to neutron emission. A recent measurement using neutron spectroscopy [8] has placed the energy of the first excited state at 2.8(1) MeV. This study employed a two-proton knockout reaction from ^{26}Ne to observe a decay of 45(2) keV from excited ^{23}O to the ground state of ^{22}O.

The present results shown in Figure 4.18 also indicate a resonance at a very low decay energy. The fit shown in the figure corresponds to a Breit-Wigner resonance with $E_{\text{decay}} = 45$ keV and $\Gamma_{\text{decay}} = 5$ keV, consistent with the resonance parameters in Ref. [8]. Figure 4.18 shows a level scheme of ^{23}O, including the decay of the first excited state to the ground state of ^{22}O.

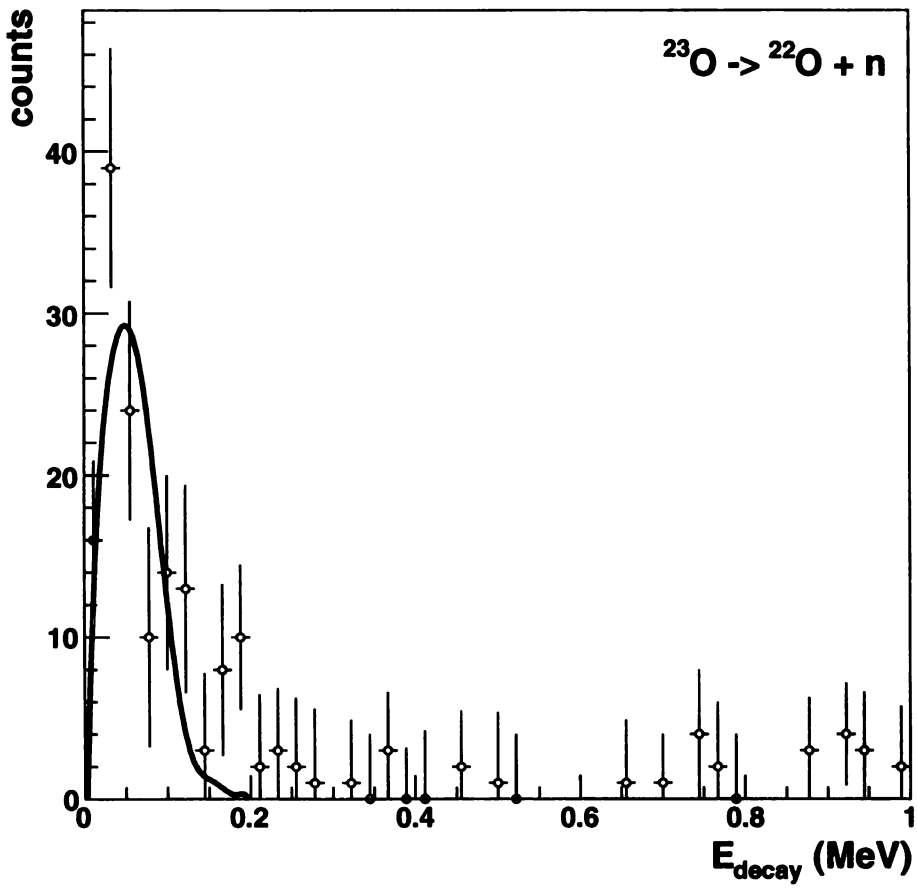


Figure 4.18: Background subtracted decay energy spectrum (open circles), with results from Monte-Carlo simulation (solid line). The simulation curve is the result of an Breit-Wigner distribution at $E_{\text{decay}} = 45$ keV, $\Gamma_{\text{decay}} = 5$ keV.

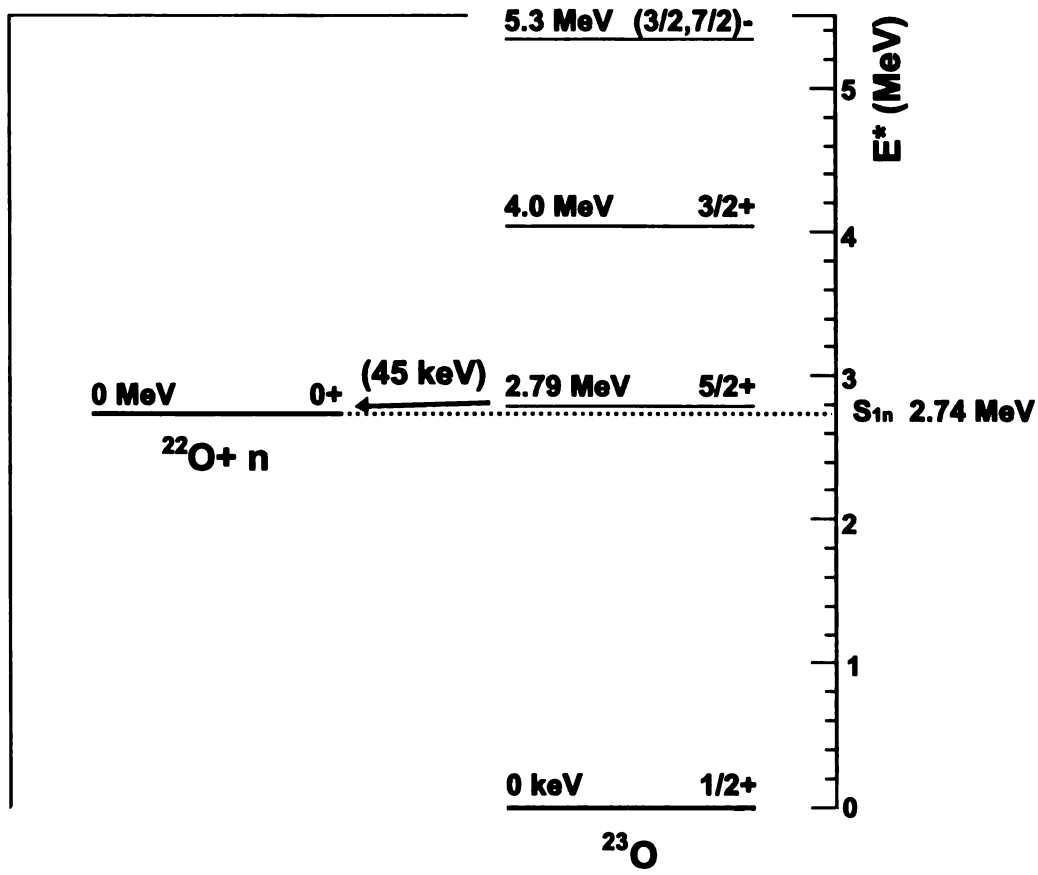


Figure 4.19: Level scheme showing the measured decay from ^{23}O to ^{22}O. Levels for the two highest excited states in ^{23}O come from Ref. [50], and the lowest lying state in ^{23}O is from Ref. [8]. Levels in ^{22}O are taken from Ref. [42].

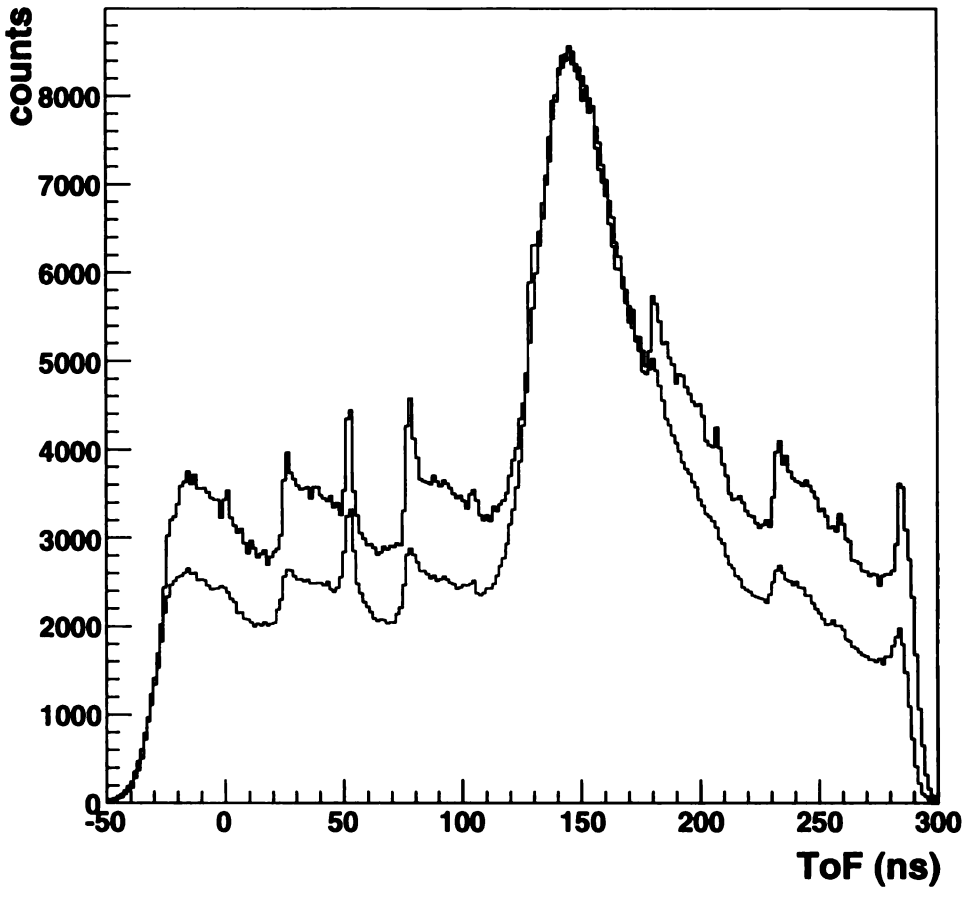


Figure 5.1: Neutron ToF for normal (blue) and high rate (black) runs.

Chapter 5

Future Prospectives

The present experiment reaches nuclei beyond the dripline only for the lightest elements (^{10}Li and ^{13}Be). Figure 3.5 shows that for heavier elements the populated nuclei are significantly removed from the dripline. In order to observe decays of more neutron-rich nuclei, the magnetic rigidity of the sweeper magnet was increased to $B\rho = 3.14$ Tm. This corresponds to a shift of about one isotope for neon. The cross section to populate these nuclei decreases, so the primary beam intensity had to be increased to compensate, and this higher beam rate increased the random background rate of neutron events. Figure 5.1 shows the neutron ToF for one of the production runs for the data shown in Section 4.1 (blue line) and a high-rate run (black line). The periodic features of the random background correspond to the cyclotron frequency. The increasing random-to-real ratio makes it more difficult to extract the signal from unbound resonances. A shift of only one isotopes towards the dripline already increases the background rate by about 40%. Thus it was impossible to reach the dripline for the heavier elements in the current experiment.

In the future SNDS could be extended to higher decay energies as well as heavier masses. The angular acceptance could be increased by substituting the quadrupole with the S800 spectrograph [51]. The target could be placed immediately in front of the sweeper magnet which would eliminate the shadowing of MoNA due to the small

bore of the quadrupole. The increased flight path and higher resolution of the S800 will also enable isotopic separation up to significantly heavier elements.

However, it does not appear to be possible to populate and measure the decay of neutron unbound states closer to the neutron dripline with SNDS in the near future. The small cross section for the production of the dripline nuclei with stable beams requires large beam intensities which in turn generate a large number of neutrons from unrelated reactions. In the longer term SNDS could be used at the next generation radioactive beam facilities [52]. The unbound states could then be populated by intense secondary beams closer to the dripline nuclei of interest.

Chapter 6

Summary

The method of sequential neutron decay spectroscopy (SNDS) has been applied for the first time using medium mass projectiles (^{48}Ca). The method succeeded in populating and identifying unbound ground states of ^{10}Li and ^{13}Be as well as unbound excited states in ^{11}Be and ^{23}O. The extracted resonance parameters were consistent with values reported in the literature. The common feature of these states is their small decay energy (<100 keV). The small angular acceptance of the experimental setup for neutrons limited the observation to unbound states with small decay energies. Isotopic separation could be achieved up to neon and was limited by the ToF resolution for the charged particles.

The future study of neutron unbound states with higher decay energies will require significant modifications of the experimental setup. The measurement of masses of nuclei beyond the dripline using SNDS will have to wait for the next generation of radioactive beam facilities.

Appendix A

Using the MoNA Simulation Code

A.1 Introduction

This document explains how to run simulations of the MoNA-Sweeper detector setup using the "simple-track" code written by H. Scheit. The code is written in C++ and uses Monte Carlo methods to simulate the influence that geometrical acceptance cuts and finite detector resolution have on a measured decay energy spectrum. This allows simulation results to be compared directly to experimental data. The simulation can also be useful in determining how the system will respond to experimental setup changes, such as changing the magnetic field of the sweeper, adding a blocker or changing the distance of MoNA from the target. This information can be very useful when planning for a new experiment as it allows the optimal running conditions to be estimated ahead of time.

The simulation starts with an incoming beam with a gaussian distribution of position, angle and energy. The beam is then stripped of a set number of protons and neutrons to arrive at an excited nucleus, which then emits a neutron according to a set decay model. The resulting fragment is then forward tracked through the sweeper magnet to its position at CRDC1, and the emitted neutron is tracked to its position in the MoNA detector. A `.root` file is then written containing the energy, position

and angle of the incoming beam; energy, position, angle, and time of the fragment at the target and at CRDC1; and energy, position, angle, and time of the neutron at MoNA. This file can then be analyzed to find the *reconstructed* target position, angle, and energy of the fragment, along with the reconstructed decay energy curve.

A.2 Getting Set Up

To start, log into a 64-bit `fishtank` machine (`lobster`, `shrimp`, `mussel`, `oyster`, `octopus`) with the account from which you plan to run the simulation. Note that you must run from one of the 64-bit `fishtank` machines and not one of the 32-bit `spice` machines.

In order to run the simulation, you first need to set some new environment variables for your user account. This will allow the simulation program to source and link with the correct libraries when you are compiling or running the program. If you plan to run the simulations as user `mona`, then these environment variables are already set. Otherwise if you are running from your own account, the environment variables are not set by default.

To set up your needed environment variables, `cd` to your home directory and open up the `.bashrc` file. Scroll down to the bottom and add the following lines:

```
export ROOTSYS=/projects/proj6/mona-sim/soft/root/root_v5.12_00/
export PATH=$ROOTSYS/bin/:$ROOTSYS/include/:$PATH
export LD_LIBRARY_PATH=$ROOTSYS/lib/:$LD_LIBRARY_PATH
```

Then save the file and type:

```
> source ~/.bashrc
```

at the command prompt. You will now have the proper environment variables set in your current session as well as any future ones that you start. This also "installs" the MoNA collaboration version of ROOT for use on the `fishtank` machines.

Once you have the environment variables set, you need to set up a new directory from which to run the simulation. The simulation program is managed with the Subversion (SVN) repository, and you should start by checking out the latest version. To do this, first create a new directory to fill with the simulation files. It is suggested that you make use of the **projects** space set aside for the MoNA simulation, which is located in **/projects/proj6/mona-sim/**. Go there and create your new directory; then **cd** to it and type:

```
> svn --username mona co http://www.mpi-hd.mpg.de/cbsvn/st/branches/mona/0.1
```

with the password **mona** (if you are prompted for one). Once this process is done running, type **ls**, and you should see a new directory called **0.1** This directory contains all of the source code for the simulation program. You still need to compile the program, however, so do:

```
> cd 0.1/
> make
```

This will create two new executables, **st_mona** and **mona_analysis**, which are the simulation and analysis programs, respectively. There is still another executable, **st_geant**, which needs to be created; to compile this do:

```
> cd st_geant
> ./compile.sh
```

It is also suggested that you make use of the **/evtdata** space set aside for mona simulations, as the files produced by the simulation program can get rather large, especially as the number of simulated events increases. To set this up, type the following commands to create your own directory in **/evtdata**:

```
> cd /evtdata/mona-sim-data/
> mkdir mysimdata
```

Then **cd** back to your **0.1** directory and make a link to the **/evdata** space:

```
> ln -s /evtdata/mona-sim-data/mysimdata/ .
```

Whenever you run the simulation program, save the output files in the **mysimdata** directory.

A.3 Running the Simulation

Once you have set up your directory as described above, you are ready to begin running simulations. All commands described from here on should be run from your new `0.1` directory unless otherwise specified. The simulation program is run from the command line and uses flags to set input and output variables. To run the simulation program, one simply types:

```
> ./st_mona [-flags]
```

where **[-flags]** represents whichever flags you want to use in running the simulation.

To get a comprehensive listing of the available flags, type:

```
> ./st_mona -?
```

The three most important flags are **-exp**, **-n**, and **-f**; these flags set the experiment number, number of simulated events to be run, and output file, respectively. The **-exp** flag is followed by a string that represents a certain experiment number, and by invoking this flag you set all other variables to their default value *for that experiment.* The **-exp** flag should always be the first one typed after **st_mona**. Setting a given **-exp** flag also instructs the program to use the proper forward COSY matrix for that particular experiment.

The **-n** flag should be followed by the integer number of simulated events that you wish to run. A suggested number which is a good compromise between run time, disk space and statistics is 100,000 events. You may wish to run more events than this if you want to eliminate statistical fluctuations in your output or if your simulation will

have large acceptance cuts, but anything more than 1,000,000 events is prohibitively slow and should typically only be used if you want to make "curve-like" histograms for publication. The default number of events run if you omit the `-n` flag is one.

As mentioned above, the filename is set using the `-f` flag, followed by the name of your output file. The output file needs to have the `.root` extension, as the simulation stores its output data in a ROOT file.[1] If you forget to set the `-f` flag, the default output filename is `st_mona.root`.

As an example, if you want to run a 100,000 event simulation for the experiment number 03038, with all parameters set to the 03038 experiment defaults, and write out the results to a file called `Sim-03038.root`, then you would type:[2]

```
> ./st_mona -exp 03038 -n 100000 -f Sim-03038.root
```

If you would like to see what the default parameters for the experiment number you are running, then you will need to view the `st_mona.cc` file:

```
> less ./st_mona.cc
```

Scroll down through the file until you find the lines that look like this:[3]

```
else if ( x == "03038") {  // Kiss 7He
  INFO("Using Default Values For Experiment 03038 (Kiss 7He)");
  eBeam           = 40.8;  //\Beam Energy in MeV/u.
  beamA           = 8;     //\Beam Mass in amu.
  beamZ           = 3;     //\Beam Charge Number
  dTarget         = 192.2; //\Target thickness in mg/cm^2.
  dEbeam          = 0.01;  //\Sigma of beam energy in percent.
  resTime         = .3;    //\Resolution (gaussian sigma) of time measurements.
  resTargetX      = 1.;    //\Resolution (gaussian sigma) of target X-position.
  resCRDC1XY      = 1.;    //\Resolution (gaussian sigma) of CRDC positions.
  resCRDC1ThetaXY = 1.;    //\Resolution (gaussian sigma) of CRDC angles.
  resMonaX        = 1.;    //\Resolution (gaussian sigma) of MoNA X-position.
  resMonaBar      = 1;     //\Turns on/off MoNA bar discreetness (1=ON).
  nNeutr          = 0;     //\Number of neutrons removed from beam.
  nProt           = 1;     //\Number of protons removed from beam.
  bSpotDx         = 0.004; //\Sigma of the beam position in the X-direction.
```

[1] It is also possible to have the simulation write out to a gsl ntuple instead of a ROOT file; to do this, give the filename the .gsltup extension.

[2] Note that the order of the flags, EXCEPT FOR THE -exp FLAG, does not matter.

[3] All positions are in meters and all angles are in radians unless otherwise specified.

```
  bSpotDy          = 0.008; //\Sigma of the beam position in the Y-direction.
  bSpotDtx         = 0.011; //\Sigma of the beam angle in the X-direction.
  bSpotDty         = 0.005; //\Sigma of the beam angle in the Y-direction.
  bSpotCtx         = 0.0;   //\Centroid of the beam angle in the X-direction.
  bSpotCx          = 0.0;   //\Centroid of the beam position in the X-direction.
  bSpotCty         = 0.0;   //\Centroid of the beam angle in the Y-direction.
  bSpotCy          = 0.0;   //\Centroid of the beam position in the Y-direction.
  crdc1MaskLeft    = 0.086; //\CRDC1 blocker position on the positive-X side.
  crdc1MaskRight   = -0.15; //\CRDC1 blocker position on the negative-X side.
  crdc2dist        = 1.00;  //\Distance from CRDC1 to CRDC2.
  monaDist         = 8.208; //\Distance from the target to the center of the
                            //\first bar of MoNA.
}
```

The comments set off by //\ in the lines above explain what each of the variables represent. Note that the **nNeut** and **nProt** variables are the respective number of neutrons and protons stripped from the beam to get to the *excited* nucleus, which always has one more neutron than the fragment that you see in your particle detectors. So for example, if you were to simulate a ^{48}Ca beam going to excited ^{23}O, and then the exited ^{23}O emitting a neutron to become ^{22}O + n, you would set **nNeut** and **nProt** to 12 and 13, respectively.

In addition to the experiment specific defaults, there are also a number of global defaults; the most important one is the excitation energy model, which is the type of lineshape that describes the decay energy of the emitted neutron. The default is set to be a uniform distribution between 0 and 5 MeV. There are a variety of excitation lineshapes available, and the decay model can be changed using the **-e** flag as described below.

A.4 Using Flags

As has been mentioned previously, nearly any input variable can be changed by using the proper flags. This is very useful when running simulations as it allows parameters to be adjusted "on the fly" to quickly see how they affect the overall measurement. The help invoked by running **st_mona -?** (or **st_mona --help**) does a good job explaining what each flag does. For reference the help output is reproduced here (some extra comments have been added in square brackets, for more clarity):

```
usage:

 -v    Run verbose
 -n    Set number of events
 -f    Set output file name.  Extension determines file type.
          root   -> ROOT file
          gsltup -> GSL n-tuple (also a file *.gsltup.dsc is written, describing
                    the columns)
 -geant <filename>  Cause st_mona to write out neutron energy & angles to an
                    ASCII file called filename
 -reac Set reaction model
         glaub (normal fragmentation w/ glauber kick)
         2body <Q-value> <Q-value spread> (two-body stripping)
                              n.b. spread is relative, e.g. 0.1 = 10% spread
 -e    Set decay energy model and parameters (MeV)
         const   <energy>          [delta-function spike at <energy>]
         uniform <e-low> <e-high> [uniform distribution]
         e1      <e-low> <e-high>
         therm   <temp>            [thermal with ''temperature'' <temp>]
         gauss   <centr> <sigma>   [gaussian]
         bw      <centr> <gamma>   [breit-wigner]
         asymbw  <energy> <RedWidth> <angMom> [energy-dependent breit-wigner]
 -be   Set beam energy per A
 -bZ   Set beam Z
 -bA   Set beam A
 -dT   Set target thickness
 -dbe  Set relative beam energy spread (sigma)
 -rTx  Set resolution of target-x
 -rt   Set time resolutions (ns)
 -rxy  Set resolution of CRDC1 xy (m)
 -rth  Set resolution of CRDC1 theta xy (rad)
 -rMx  Set resolution of MoNA X
 -rMyz Switch MoNA Y and Z discretization on/off (1/0)
 -dKE set Kinetic Energy loss/gain from the reaction (in MeV/u).
 -strag scale
 -glaub scale
 -np   number of removed protons
 -nn   number of removed neutrons
 -md   distance to MoNA (middle of A8)
 -cmL  position of left  edge of blocker (before CRDC1)
 -cmR  position of right edge of blocker (before CRDC1)
 -c2d  distance between CRDC1 and CRDC2 (meters)
 -ctx  centroid of x angle in radians
 -cx   centroid of x position in meters
 -cty  centroid of y angle in radians
 -cy   centroid of y position in meters
 -dtx  sigma of x angle (rad)
 -dx   sigma of x posn (m)
 -dty  sigma of y angle (rad)
 -dy   sigma of y posn (rad)
 -disType type of MoNA X distribution
 -exp  <exp number>  set default values for experiment <exp number>
          03033 - Nathan's 230, 220
          03038 - Kiss 7He
.....................................................................
```

Unless otherwise specified, whenever you invoke a flag, it should be followed by a number representing its argument(s).

As an example, if I want to run a simulation of the 03038 experiment, with the following changes made to the default input parameters:

- Decay energy = Breit-Wigner with $E_{decay} = 0.74$ MeV, $\Gamma_{decay} = 0.1$ MeV
- Beam energy = 42.0 MeV/u with 1% spread
- Centroid of beam x position = 0.005 m
- Centroid of beam x angle = 0.002 rad
- Centroid of beam y position = 0.004 m
- Centroid of beam x angle = 0.002 rad
- Sigma of beam x position = 0.003 m
- Sigma of beam x angle = 0.001 rad
- Sigma of beam y position = 0.002 m
- Sigma of beam x angle = 0.003 rad
- Resolution of CRDC positions = 0.004 m,

then I would type the following:

```
./st_mona -exp 03038 -n 100000 -f Sim-03038.root -e bw 0.74 0.1 -be 42.0 \
-dbe 0.01 -cx 0.005 -ctx 0.002 -cy 0.004 -cty 0.002 -dx 0.003 -dtx 0.001 \
-dy 0.002 -dty 0.003 -rxy 0.004
```

Doing this will give me a new file called `Sim-03038.root`, with the input parameters set as above.

The following subsections explain some of the more commonly used flags in more detail; if the reader is simply interested in learning how to run the simulation, then he/she can safely skip to section A.5

A.4.1 Decay Energy Model

As mentioned above, the decay energy model can be set using the `-e` flag. The most common types of decays are the Breit-Wigner, energy-dependent Breit-Wigner, and Thermal. The Breit-Wigner and energy-dependent Breit-Wigner represent the two most common lineshapes for unbound resonances, and they are respectively invoked with the `bw` and `asymbw` arguments to the `-e` flag. The `bw` argument expects two numbers to follow it, representing (in order) central Breit-Wigner energy, and the width of the lineshape; the `asymbw` argument is followed by three numbers representing (again, in order) central energy, *reduced* width, and angular momentum. The thermal distribution is characteristic of a neutron background, and is invoked with the `therm` argument to the `-e` flag; `therm` is followed by one number representing the "temperature" of the thermal distribution, in MeV. Typical temperatures are usually on the order of 1 MeV.

A.4.2 Stripping Reaction Model

The stripping reaction model can be changed by invoking the `-reac` flag. The global default[4] reaction is a simple Glauber model, which conserves the momentum of the incoming beam. This model allows for the option of setting a momentum kick and angular straggling scale, using the `-glaub` and `-strag` flags, respectively. The `-glaub` and `-strag` flags are each followed by one number which represents the overall scale for each parameter; the default value for both `-glaub` and `-strag` is 1.

A second stripping reaction model has also been added as it produces a better fit to the data for the 03038 ^{7}He experiment. This is a "two-body" stripping reaction, which conserves total energy as opposed to total momentum. It can be invoked by following the `-reac` flag with the argument `2body`; the program will then expect two numerical arguments representing (in order): Q-value of the stripping reaction (in

[4]Note that since the Glauber model is the default reaction, you do not have to invoke the -reac flag at all if you wish to use it.

MeV) and spread in the Q-value (in percent). So for example, if you wanted to model your stripping reaction as a two-body reaction with $Q = -8.0$ MeV, $dQ = 1\%$, then your command to run **st_mona** should include:

```
-reac 2body -8.0 0.01
```

A.4.3 GEANT Output

It is possible to have **st_mona** write out neutron energy and angles into a text file which can then be read into a GEANT simulation to more accurately model the interaction of the neutrons in MoNA. To do this, use the **-geant** flag, followed by the name of the text file that you wish to write out, e.g.

```
-geant geant_out.txt
```

Note that invoking the **-geant** flag will not affect the rest of the simulation program; neutrons will still be tracked to MoNA as usual.

As mentioned above, the text file written by using the **-geant** flag can be read into GEANT to do a more accurate simulation of the neutron interaction;[5] the output of the GEANT simulation is another text file, which can be integrated into your existing simulation data using the **st_geant** executable. This program is located in the **st_geant** subdirectory of your **0.1** directory. It takes in two files: a **.root** file produced by **st_mona**, and a text file produced by GEANT, and merges the data into a new **.root** file. This new file links the neutron events produced by GEANT with the charged fragment events produced by **st_mona**, and it can be analyzed just like a normal **st_mona** output. For more information on how to run the **st_geant** program, go into the **st_geant** directory and type **st_geant --help**.

[5]Running the GEANT simulation is outside the scope of this manual; for information on how to do this, contact Artemis Spyrou.

A.5 Analyzing Your Simulation

The `.root` file produced as described in Section A.3, contains only simulated target, CRDC1, and MoNA parameters; it does not include the inverse reconstructed parameters necessary to compare simulation results directly with data. These inverse reconstructed parameters can be calculated by using the program **mona_analysis**. This program has two basic modes: it can either be used to calculate the needed reconstructed parameters directly, with the results written out to a `.root` file, or it can be used to convert the `.root` file produced by **st_mona** into a SpecTcl filter file which can then be analyzed using the **SpecTcl_filter** program available on **zuma.iusb.edu** or on the NSCL **spice** machines.

To use **mona_analysis** to convert a `.root` file (called, e.g. **simfile.root**) into a filter file, simply type:

```
> ./mona_analysis -flt -if simfile.root -of simfile.flt
```

Here the `-flt` flag tells the program to convert the `.root` file into a SpecTcl file; the `-if` flag sets the name of the input file, and the `-of` flag sets the name of the output file. Note that the output file when using the `-flt` option must have the extension `.flt`.

If you wish to use **mona_analysis** to analyze your simulation file directly, then the procedure is to run the program as follows:

```
> ./mona_analysis -if simfile.root -of simfile_ana.root -frag 6He
```

Doing this would run **mona_analysis** with on an input file called **simfile.root** (set with the `-if` flag), and it would return a new file, **simfile_ana.root** (set with the `-of` flag). The **-frag** flag tells the program what type of analysis to do; it sets the inverse map to be used; the mass, charge, and brho of the fragment being reconstructed; and the target thickness and material. So in the example above, the program would perform the reconstruction using the parameters which are defined for the **6He**

fragment. Note that the argument of the **-frag** flag must be a string which is defined within the **mona_analysis** program; to see which strings are available, and the settings used for each string, view the **mona_analysis.cc** file, and scroll down until you see some lines that look like this:

```
else if (frag == "6He") {
  INFO("Using settings for fragment 6He");
  e->setMaps(m6He, m6Hei);
  e->fragA = 6;
  e->fragQ = 2;
  e->dTarg = 188;
  e->targA = 9;
  e->targZ = 4;
}
```

Looking at the lines above, you can see that the **"6He"** flag sets the reconstructed fragment to be ^{6}He; target thickness to be 188 mg/cm^{2}; and target material to ^{9}Be. It also tells the program to use the inverse map named **m6Hei**; the meaning of this map name will be explained in Section A.6.

Once you have run **mona_analysis**, you now have a full set of simulated and reconstructed parameters available to you, contained in the output files of **st_mona** and **mona_analysis** (e.g. **simfile.root** and **simfile_ana.root** in the example above). These files can be analyzed using the ROOT data analysis program. If you are unfamiliar with ROOT, some basic instructions for using it can be found in Appendix B.

One important consideration to keep in mind is that the simulation and analysis programs track and store every event, even those which are "unphysical," such as events for which the neutron does not hit MoNA or events for which the fragment does not hit CRDC1. The program does flag these events by setting the kinetic energy of the offending particle to zero. So for example, in an event for which the neutron does not hit MoNA, the parameter representing neutron kinetic energy will be set to zero, and likewise, for an event where the fragment misses CRDC1 (or hits the blocker),

the fragment kinetic energy parameter is set to zero. Whenever looking at simulation results as a comparison to experimental data, the simulation spectra should always be gated such that events with $KE_{frag} = 0$ or $KE_{neutron} = 0$ are excluded. Depending on the geometry of your individual setup, you may need to make other cuts as well; for example the simulation does not flag events for which the fragment hits CRDC1 at a large enough angle that it misses CRDC2. If you want to exclude those events, then you need to calculate CRDC2 position based on the simulated position and angle at CRDC1 and then apply the physical cuts directly.

A.5.1 Parameter Names

This section explains the names of parameters stored in the `.root` files created by `st_mona` and `mona_analysis`. It assumes a basic level of knowledge of the ROOT program. Readers who are not experienced in ROOT should read Appendix B first and then return to this section.

The `.root` files output by `st_mona` and `mona_analysis` each contain an incomplete set of parameters; the `st_mona` file only contains simulated and forward-tracked parameters, while the `mona_analysis` file only contains reconstructed parameters. In order to view and compare simulated and reconstructed parameters, you will need to use the `AddFriend` command available in ROOT.

Parameters in a file created by `st_mona` are stored in a `TTree` called `t`, and they are named according to a system which denotes beamline position, particle number (e.g. fragment or neutron), and parameter name. The system is best explained with an example: the parameter denoting CRDC1 x-position is `b7p0x`, meaning that we are at beamline position 7 (crdc1), looking at particle 0 (the charged fragment), and the final `x` tells us that we are looking at x-position. The available parameter numbers are 0 (charged fragment) and 1 (neutron). The beamline numbering scheme is described in Table A.5.1, and the parameter naming scheme in Table A.5.1.[6] For a few more

[6]Each of the parameters listed in Table A.5.1 is not necessarily available for every beamline

Number	Location
1	Before Target
2	After Target
7	CRDC 1
13	MoNA

Table A.1: Numbering scheme for beamline elements as produced by the program st_mona.

Name	Parameter
Ekin	kinetic energy (MeV)
x	x-position (m)
y	y-position (m)
tx	x-angle (rad)
ty	y-angle (rad)
z	z-position (m)
t	time (ns)

Table A.2: Naming scheme for simulation parameters as produced by the program st_mona.

examples, consider the following:

b1p0Ekin – Beam energy before the target.

b2p0tx – Fragment x-angle after the target.

b13p1t – Neutron ToF to MoNA.

Parameters for analyzed **.root** files are stored in a **TTree** called **at**. Because the analyzed file does not contain parameter values at a variety of beamline positions, it uses a more straightforward parameter naming scheme, as described below:

deltaE[1] – Energy loss in the target.

exen – Decay energy.

tLab – Opening Angle betwen fragment and neutron.

vRel – Relative velocity betwen fragment and neutron.

nVel – Neutron velocity.

element. To get a full listing of the available parameters, from within ROOT, use the t.Print() command. There are also a few parameters specific to b2 which are not included in Table A.5.1; these are: TP (position of the interaction within the target), dE (energy loss in the target), and R_exen (excitation energy).

fVel – Fragment velocity.

fragKinE – Fragment Kinetic Energy.

neutronKin – Neutron Kinetic Energy.

nTheta – Neutron θ.

nPhi – Neutron ϕ.

fTheta – Fragment θ.

fPhi – Fragment ϕ.

fpATA – Fragment x-angle at target.

fpBTA – Fragment y-angle at target.

fpYTA – Fragment y-position at target.

fYfp – Fragment y-position at CRDC1.

fBfp – Fragment y-angle at CRDC1.

fXfp – Fragment x-position at CRDC1.

fAfp – Fragment x-angle at CRDC1.

There are more parameters present in the **at** tree, but they are not needed for a typical user.

A.6 Customizing the Simulation

It is likely that at some point you will need to add code to the simulation in order to continue with your analysis. Since the code is managed with SVN, modifying it is a fairly low-risk task, even for those who are inexperienced in C++. As long as the user does not "check in" his/her changes, the worst case scenario of a bad modification is that the code in the user's directory has to be scrapped, and the current version re-checked out from the repository.

The most common type of modification is to add a new experiment option to the simulation. In order to do this, you need to modify three files: **st_mona.cc**,

mona_analysis.cc, and mk_maps_icc.sh. The first file that you should modify is st_mona.cc. Open up this file and scroll down until you see lines that look like:

```
else if ( x == "05039") {  // Hoffman 250
      INFO("Using default values for experiment 05039 (Hoffman 250)");
      eBeam           = 84.0;
      beamA           = 26;
      beamZ           = 9;
      dTarget         = 500.0;
      dEbeam          = 0.025;
      resTime         = 0.3;
      resTargetX      = 0.0007;
      resCRDC1X      = 0.0013;
      resCRDC1ThetaX = 0.0013;
      resCRDC1Y      = 0.0013;
      resCRDC1ThetaY = 0.0013;
      //resMonaX1        = 0.04;
      //resMonaX2       = 0.10;
      //resMonaP        = 0.66;
      //resMonaBar      = 1;
      nNeutr          = 0;
      nProt           = 1;
      bSpotDx         = 0.005;
      bSpotDy         = 0.004;
      bSpotDtx        = 0.009;
      bSpotDty        = 0.0035;
      bSpotCtx        = 0.008;
      bSpotCx         = 0.00;
      bSpotCty        = -0.001;
      bSpotCy         = -0.001;
      crdc1MaskLeft   =  0.15;
      crdc1MaskRight  = -0.15;
      crdc2dist       = 1.88;
      monaDist        = 8.20;
     }
```

Copy and paste these lines to create a new experiment. In the new lines, replace the 05039 in the line `else if ( x == "05039")` with the name that you want for your new experiment. You should also change the string in the line beginning with `INFO` to something sensible for your new experiment. All of the other variables should be set to their proper values for your experiment. An explanation of what each variable represents is given in Section A.3.

The next lines that you should edit look something like this:

```
  else if ( beamA - nProt - nNeutr == 25  && \
            beamZ - nProt == 8 && beamZ == 9) {
    dipole = new StPropMapCosy(m240);
    dmona = new StPropDrift(monaDist - m240.getLen());
    INFO("using 24o map for 05039");
  }
```

As before, copy and paste these lines, and change the line reading

```
beamA - nProt - nNeutr == 25  && beamZ - nProt == 8 && beamZ == 9)
```

to reflect your new number of total particles lost, number of protons lost, and beam Z. Recall that the final mass number should be that of the excited nucleus, *before* emitting the final neutron. You also need to change the map name in the second and third lines. In the example shown above, the map name is `m240`, so you would replace `m240` with your new map name. At this point the map name can be anything you want; you will give it a meaning later when you modify `mk_maps_icc.sh`. Finally you should change the string coming after `INFO` to reflect the new map and experiment.

The final change to make to `st_mona.cc` is to update the help message to include your new experiment. This is not necessary for the code to run properly, but doing it will make everyone's life a little easier. Scroll up near the top of the file until you find the lines:

```
" -exp  <exp number>  set default values for experiment <exp number> \n"
"          03033 - Nathan's 230, 220\n"
"          03038 - Kiss 7He\n"
"          03048a - 12Be g.s. to 10Be\n"
"          03048b - 11Be Coul to 10Be\n"
"          05039  - Hoffman 250\n"
.....................................................................
```

Pick a line to copy and paste, and update the `<exp number>` and description fields to reflect your new experiment.

The next file that you should edit is `mona_analysis`. Only one set of lines needs to be updated here: those that look like:

```
else if (frag == "240") {
  e->setMaps(m240, m240i);
  e->fragA = 24;
  e->fragQ = 8;
  e->dTarg = 500;
  e->targA = 9;
  e->targZ = 4;
}
```

Copy and paste these lines, and then change the text reading `frag == "240"` to reflect your new fragment name (in this example, you would replace 240 with the new name. The new name can be anything you like (but it must be unique), and it is what follows the `-frag` flag when you run `mona_analysis`, as described in Section A.5. Next you should update the line reading `e->setMaps(m240, m240i);` This line is what sets the names of the forward and inverse maps. The name of the forward map, which in this example is `m240`, must match the new map name that you set in `st_mona.cc`. The name of the inverse map—`m240i`—can be anything you want since it has not yet been set elsewhere, but it is suggested that you stick to the convention of simply adding an `i` to the end of the forward map name. Finally you will need to update the remaining lines to reflect the correct variable values for your experiment; the meaning of each variable is explained in Section A.5.

The last file that you need to edit, `mk_maps_icc.sh` is what ties together the map names that you set in `st_mona.cc` and `mona_analysis.cc` to the actual COSY map files. In the `mk_maps_icc.sh` file, copy and paste the lines that look like:

```
echo "// forward map"
mkMapEntry m240   24  8  1.5741  3.77548  24o-cosy.map
echo "// partial inverse map"
mkMapEntry m240i  24  8  1.5741  3.77548  24o-sch.imap
```

You need to edit the two lines beginning with `mkMapEntry`. These two lines set variables for the forward map (top line) and inverse map (bottom line). The expla-

nation of each entry in these lines is given below:[7,8]

`m240` and `m240i` – Name of the forward and inverse maps, as assigned in `st_mona.cc` and `mona_analysis.cc`.

24 – Mass of the tracked/reconstructed fragment (the fragment that is left *after* neutron emission).

8 – Charge of the tracked/reconstructed fragment.

`1.5741` – Central track distance from the target to CRDC1.

`3.77548` – Brho setting of the map file that you are using.

`24o-cosy.map` and `24o-sch.imap` – Filenames of your COSY forward and inverse map files.

Once you have edited these lines and saved all of your files, you should be ready to recompile the programs. To do this, from within the `0.1` directory, type **`make`**, and if there are no errors you should end up with new executables that include your changes.

A.7 Using SVN

If you have made modifications to your simulation code that you think might be useful to the rest of the collaboration, then you should check these changes into the respsitory so that others can access your updated code. It is also a good idea to regularly update your own directory so that your code always contains the latest implementations. Below are some of the basic SVN commands that you will need:

`> svn update` – Update to latest code version.

[7]Note that both the forward and inverse map files must be in "COSY format" (without line headers) and not "SpecTcl format" (with line headers). If need to convert a map file to COSY format, use the program spctkl2cosy, which is located in /projects/proj2/sweeper/bin/. This program will only run on 32-bit spice machines and not the 64-bit fishtank machines.

[8]It is recommended that you DO NOT follow the past practice of copying individual map files into the 0.1 directory, as doing this leads to clutter and compatibility issues between different versions. A better method is to simply include the full filepath of the map file in its original location, e.g. /projects/proj2/sweeper/maps/.../*.map

> **svn add <filename>** – Add a file to the repository.

> **svn ci <filename>** – Check in file to SVN (must be added first).

> **svn co <filename>** – Check out file from SVN.

> **svn stat** – View status of all SVN files.

As explained above, if you want to check in your changes to the repository, use the command **svn ci <filename>**, where **<filename>** is the name of the file that you are checking in (you can also use **svn ci** without the **<filename>** argument to check in all files in your directory). When you use the **svn ci** command, a **vi** editor will automatically open up to allow you to make a comment on the changes that you are adding. To begin inserting text, press **i**, and when you are done writing your comment, hit the **esc** key, followed by **:wq** and then hit the **Enter** key.

Before you check in any changes, your new code should be well tested to ensure that there are not any bugs. You should also be sure that your new code does not cause any incompatibilities with the existing code. This can be checked using the **svn stat** command, which lists all edited files in the directory with a status character before the filename. The status characters are as follows:

M–modified (not checked in yet)

A–added

?–unknown (not added yet)

C–conflict (please edit to agree with checked out version)

U–updated

If one of your files is flagged as having a conflict, you must edit the file until the conflict is resolved. Once you have done this, type:

> svn resolved <filename>

to tell subversion that you have resolved the conflict. For more information on using SVN, see the documentation at: **http://subversion.tigris.org/**.

Appendix B

Using ROOT

B.1 Introduction

ROOT is an "object oriented data analysis framework" written by programmers at CERN. It is particularly suited for analysis of the large data sets found in nuclear physics. It is capable of performing basic analysis tasks such as gating, histograming and fitting, in addition to much more complex tasks. ROOT is primarily command line driven, using the CINT C/C++ interpreter.

This manual is intended to introduce those who are new to ROOT to some of its basic functions. After reading this manual you should have a "SpecTcl like" level of functionality in using ROOT, i.e. you should be able use ROOT perform all of the tasks that are possible in SpecTcl (plus a few more).[1] At times the manual may gloss over some of the more subtle subjects involved in using ROOT; this is done in the interest of promoting simplicity and quick learning of the program. The manual also makes purposeful use of imprecise terminology as opposed to technically correct C++ jargon, with the hope that it will be better understood by readers with little or no knowledge of C++.

[1] SpecTcl is just being used as an analogy here; if you aren't familiar with SpecTcl, don't worry about it.

B.2 Installing ROOT

Before you can start to use ROOT, you need to "install" it (e.g. set the needed environment variables) on your NSCL user account.[2] If you have already set up your account to run `st_mona` simulations—as outlined in Appendix A—then you are all set and can begin using ROOT on any of the 64-bit fishtank machines. Otherwise, open up your `~/.bashrc` file, and add the following lines to the end of the file:

```
export ROOTSYS=/projects/proj6/mona-sim/soft/root/root_v5.12_00/
export PATH=$ROOTSYS/bin/:$ROOTSYS/include/:$PATH
export LD_LIBRRY_PATH=$ROOTSYS/lib/:$LD_LIBRARY_PATH
```

Then save the file and type

```
> source ~/.bashrc
```

Once you have done this, the command to begin the ROOT program is:

```
> root.exe
```

When you type this at the command line, ROOT should start up and you should see something like this:

```
*******************************************
*                                         *
*        W E L C O M E  to  R O O T       *
*                                         *
*   Version   5.12/00      10 July 2006   *
*                                         *
*  You are welcome to visit our Web site  *
*          http://root.cern.ch            *
*                                         *
*******************************************

FreeType Engine v2.1.9 used to render TrueType fonts.
```

[2] If you are logged into the NSCL servers as user `mona`, then the environment variables are already set; in this case you can go ahead and begin using ROOT on the 64-bit fishtank machines.

```
Compiled on 15 November 2006 for linux with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.13, June 8, 2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

ROOT's command line works much like a BASH shell: you can scroll through your most recent commands by hitting the up-arrow key, and auto-complete recognized commands using the `Tab` key. You can quit the program by typing `.q` at the command line. Also note that all commands in ROOT are case sensitive.

B.3 Files and Trees

ROOT reads data that is stored in a binary file with the `.root` extension. These files are capable of storing many types of objects, but for the purposes of this document we will focus on the most basic one: the `TTree`. There are two example files, `example1.root` and `example1_ana.root` located in the `/projects/proj1/MoNA/ROOT/` directory. It is suggested that you use those two example files to practice the commends being presented throughout this document.

To read data from a `root` file, you must first read the file into ROOT's memory. There are two ways to this. The easiest is to simply follow the `root.exe` command by the name of the file, e.g.

```
> root.exe example1.root
```

The second way is to open up the file once you have already begun the program; you can do this using the following command:

```
TFile * _file0 = TFile::Open("example1.root");
```

One note about this method: the `_file0` can be thought of as a "variable" name (more accurately, it's the name of the pointer to the `TFile` object). It could be set to anything that you wish.

Event #	Value of P1	Value of P2
1	74.689	5670.4
2	93.452	6987.6
3	45.657	3331.4
4	22.389	3787.6
5	61.723	2229.3

Table B.1: Simple example of the type of data stored in a `TTree`.

Now that you know how to read a file into ROOT, we'll go a bit into what makes up a ROOT file. As mentioned above, the most fundamental part of a ROOT file is the `TTree`; a `TTree` is essentially a collection of "events" and parameters. Each event is a collection of data values for all of the parameters present in the file. As a simple example, consider a `TTree` that stores two parameters, P1 and P2, and five events. The structure of the data might look something like what is shown in Table B.3, with each event number being associated with specific values of P1 and P2.

Each `TTree` has a "name" associated with it, and in order to perform operations on the tree, you must know what that name is. To see the names of all the trees contained with the file you have open, use the command:

```
.ls
```

and you should see something like:

```
root [2] .ls
TFile**         example1.root
 TFile*         example1.root
  KEY: TTree    t;1     simple-track tree
```

Here the line `KEY: TTree t;1 simple-track tree` tells you that you have a `TTree` with the name `t` stored within this particular ROOT file.

The *parameters* in a `TTree` are given names by the person or program who writes the ROOT file; to see a list of all of the parameters stored in your particular `TTree`, type:[3]

[3]Throughout this document, we will use the tree name "t" to represent a generic TTree, and

```
t->Print("all");
```

doing this will show you a bunch of entries that look like this:

```
*.....................................................................*
*Br    4 :b1p0Ekin   : b1p0Ekin/D                                      *
*Entries :   100000 : Total  Size=     802866 bytes  File Size  =     697625 *
*Baskets :       25 : Basket Size=      32000 bytes  Compression=   1.15     *
*.....................................................................*
```

The lines above give information about the parameter called **b1p0Ekin**. For the purposes of this document, you just need to be able to read off the parameter name from lines like the one above: as you can see the name, **b1p0Ekin**, is located on the first row directly after the first colon.

One useful (and necessary if you want to look at MoNA simulation data) feature of ROOT trees is the ability to merge **TTrees** from separate ROOT files together, by adding the second tree as a "friend tree" to the first. Essentially, when you add a friend tree to an existing tree, the existing tree looks as if it contains all of the parameters present both in itself and in the friend tree. As as example, consider the files **example1.root** and **example1_ana.root**. Start up ROOT and load in the file **example1.root**. Now add as a friend the tree contained in **example1_ana.root"** (which is named **at**) by doing the following:

```
t->AddFriend("at=at","example1_ana.root");
```

Once you have done this, you can seamlessly work with parameters in **example1_ana.root**, just as if they were contained in **example1.root**.

In addition to the parallel merging capabilities of **AddFriend**, it is also possible to merge multiple files in series, such that a group of files can be used as if it were one large file. A grouping of files in series is called a **TChain**, and the syntax to create one is:

---

all "operations" on a TTree will assume that the tree name is "t" (unless specifically specified otherwise).

```
TChain * mychain = new TChain("t");
```

Note that the `t` in the expression above should be substituted with the actual name of the `TTrees` in the files that you are chaining together.

The `new TChain()` command shown above only creates a new `TChain`; you still need to add files to it using:

```
mychain->Add("file1.root");
mychain->Add("file2.root");
mychain->Add("file3.root");
mychain->Add("file4.root");
mychain->Add("file5.root");
mychain->Add("file6.root");
mychain->Add("file7.root");
.........................
```

Once you have done this you can treat `mychain` as if it were a "normal" `TTree` name.

B.4 Drawing Histograms

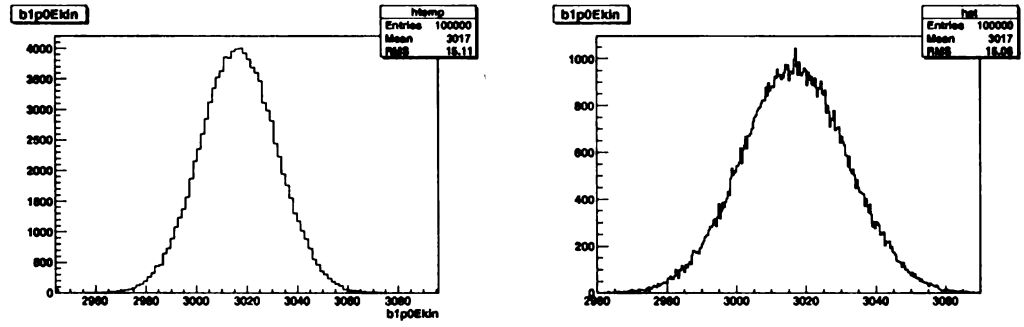
Histograms are one of the most commonly used methods of viewing data in nuclear physics. A histogram in ROOT can be thought of as the analogy of a spectrum in SpecTcl: for a given parameter it displays the number of counts falling within a user-defined bin range. The command used to draw a histogram is best illustrated with an example: lets say I am interested in seeing a histogram of the parameter `b1p0Ekin`, and I want to let ROOT set the histogram limits and bin ranges automatically. In order to do this, the command is as follows:

```
t->Draw("b1p0Ekin");
```

The histogram drawn by this command is shown in Figure B.1(a). Now lets say that I want to draw another histogram of `b1p0Ekin`, but I want to manually set it to display from 2960 to 3070 in 300 bins. The command to do this is as follows:

```
t->Draw("b1p0Ekin>>hst(300,2960,3070)");
```

which draws the histogram seen in figure B.1(b). The instructions to set the binning and ranges are contained in the characters `>>hst(300,2960,3070)`. Here `hst` is the histogram "name," and could be set to anything.[4] The 300, 2960, and 3070 represent the number of bins, low bin and high bin, respectively.



(a) (b)

Figure B.1: Example ROOT histograms; the histogram in the left panel is drawn with automatic binning and ranges, and the one in the right panel is manually set to go from 2960 and 3070 in 300 bins.

The syntax to draw a two-dimensional histogram is an extension of the 1d syntax shown above. For example, if I want to plot the parameters `b1p0Ekin` vs. `b2p0Ekin`, with `b1p0Ekin` displayed on the y-axis from 2960 to 3070 in 300 bins, and `b2p0Ekin` displayed on the x-axis from 400 to 800 in 300 bins, the command is:

```
t->Draw("b1p0Ekin:b2p0Ekin>>hst(300,400,800,300,2960,3070)");
```

Note that the syntax for specifying parameters is Y:X, while the syntax for setting bin and axis ranges is (somewhat unintuitively): (xbins, xlo, xhi, ybins, ylo, yhi).

The 2d draw command shown above will display the desired histogram using a 2d black and white scatterplot. It is usually desirable to view histograms in color, with different hues representing different z-axis values. Before you try drawing a 2d histogram in color, you should change the color scheme from the default (which is

[4] Although histogram names can be reused, doing so will delete the current histogram with that name from memory. Reusing histogram names can also lead to segmentation faults, so it is advisable to choose a unique name for each new histogram that you draw.

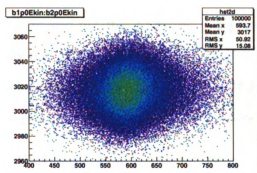
ugly and hard to decipher) to a more traditional blue-green-red. You can do this with the command:

```
gStyle->SetPalette(1);
```

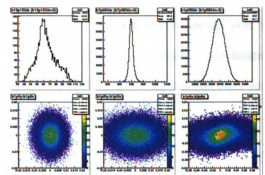
Now that you have a nice palette, the command to draw a 2d histogram in color is:

```
t->Draw("b1p0Ekin:b2p0Ekin>>hst(300,400,800,300,2960,3070)","","col");
```

It should be obvious that the `"col"` is what tells ROOT to draw the histogram in color. It should also be noted that if you want to display a color scale on the side of your histogram, you can replace `"col"` with `"colz"` in the command above. You may be a bit confused by the unfilled set of `""` in the command. These are being used as "placeholders" for the gate argument, which is described in Section B.6. The results of the command above can be seen in Figure B.2(a).



(a) Example two-dimensional ROOT histogram displaying `b1p0Ekin` vs. `b2p0Ekin`, with user defined binning of 300 bins between 400 and 800 on the x-axis and 300 bins between 2960 and 3070 on the y-axis.



(b) Example of a `TCanvas` divided into three pads in the x-direction and two pads in the y-direction.

Figure B.2: Example 2d ROOT Histogram (left panel) and Divided Canvas (right panel).

Once you have drawn a histogram using the `Draw()` command, it is saved into memory. You can access it later using the command, `histname->Draw();`, which will re-draw the histogram named `histname` in the current window.

B.5 TCanvases

As you should have noticed, using the `t->Draw()` command creates a new graphical window in which the histogram is drawn. This window is called a **TCanvas**. You can also create a **TCanvas** manually, using the following command:

```
TCanvas * c2 = new TCanvas();
```

Here `c2` is the "name" of the **Tcanvas**, and as with histogram or tree names it could be anything you want. You can have multiple canvases open during a ROOT session, but only one of the canvases is active at a given time; the active canvas will have a yellow outline around its edges indicating that it is the active one. The command to switch between canvases is `canvasname->cd();` e.g. if I want to switch to the canvas called `c1`, then I would use:

```
c1->cd();
```

It is also possible to divide a canvas up into "pads" or sections, allowing you to draw multiple histograms on one canvas. This is done using the `Divide()` command. For example if I want to divide `c1` so that it has three pads in the x-direction and two pads in the y-direction, then I would use:

```
c1->Divide(3,2);
```

Figure B.2(b) shows a 3×2 divided canvas, with an example histogram drawn in each subpad.

Another useful feature is the ability to look at histograms with axes on a log scale. The command to do this in ROOT is `SetLog*()`, where `*` is the name of the axis (x, y, or z) in lowercase. The `SetLog` command only operates on the canvas or pad that is currently selected. As an example, if I have a one dimensional histogram and want to view the y-axis on a log scale, I would use:

```
gPad->SetLogy();
```

For a two dimensional histogram for which I want the z-axis on log scale, I would type:

```
gPad->SetLogz();
```

To go back to linear scale, the command is `SetLog*(0)`, e.g. if I wanted to change the y-axis back to linear, I would type:

```
gPad->SetLogy(0);
```

It is possible to modify histograms and other graphical objects using the built in GUI. To make use of this, go to the **View** menu on your canvas and select the **Editor** option. You will see a new window appear in the left edge of your canvas. Use of the editor is fairly intuitive. Note that if you click on different parts of the canvas, the object being edited (and thus the options available in the editor) change.

As a final note, you will probably want to save pictures of your canvas to file. The command to do this is:

```
c1->Print("filename.*")
```

where * is the file extension. ROOT will automatically set the type of file being written based on the extension you select. ROOT can save canvases to many different file types, including **.pdf**, **.ps**, **.eps**, **.svg**, **.png** and **.jpg**. You can also save your canvas using the **File->Save As** menu, or you can send it directly to a printer using **File->Print**. Histograms points can also be written out to an ASCII file, using the command:

```
hstname->Print("all"); > outputfilename.txt
```

This will write out the points of the histogram into a text file called **outputfilename.txt**. The formatting of the output file is a bit clumsy; there is a way to write out histograms into a plain two column format, but it requires the use of custom commands; hence it will be explained later on.

B.6 Gates

The ability to set "gates" (logical restrictions) on parameters is needed to do any sort of nuclear physics analysis. Gates in ROOT are treated as logical C++ statements. Some of the more commonly used logical statements are: "greater than" (>), "less than" (<), "equal to" (==), "not equal to" (!=), AND (&&), and OR (||). Any combination of these logical statements can be used. The gate command is the second argument of the `Draw()` command and is surrounded by " ". As an example, if I want to view a histogram of the parameter **b7p0x**, with the restriction that another parameter, **b7p0tx**, is greater than zero, I would use the following command:

```
t->Draw("b7p0x","b7p0tx > 0");
```

As a more complicated example, if I wanted to see a histogram of **b7p0x** subject to the following conditions:

-b7p0tx greater than 0.
-b1p0Ekin less than 3000.
-b7p0Ekin not equal to zero,

then I would type the following:

```
t->Draw("b7p0x","b7p0tx > 0 && b1p0Ekin < 3000 && b7p0Ekin != 0");
```

In additional to one dimensional gates, ROOT can also create two dimensional gates which constrain the values of two parameters to lie within a certain shape. Two dimensional gates are given a name, and can be used just like any other logical statement. For example, if I want to apply a 2d gate called "my2dgate" to a histogram containing the parameter **b1p0x**, I would do:

```
t->Draw("b1p0x","my2dgate");
```

Two dimensional gates can be combined with other logical statements as well, so if I wanted to draw **b1p0x** subject to both "my2dgate" and the requirement that **b1p0tx** be greater than zero, then I would do:

```
t->Draw("b1p0x","my2dgate && b1p0tx > 0");
```

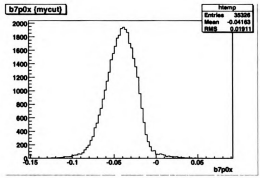
The easiest way to set a 2d gate is to make use of the **TCanvas** GUI. The steps to do this are outlined below:

1. Draw the 2d histogram.
2. In the **TCanvas** containing the new histogram, select **View->Toolbar**.
3. In the toolbar, click on the cut tool. It has a picture of scissors on it and is the farthest right tool on the toolbar.
4. Go onto the histogram and start drawing your gate. Single clicking sets a new point in the contour, and double clicking closes the gate.
5. Move the mouse cursor over the gate (the cursor will turn into a hand with a pointing index finger when you are in the right place), and right click. Then select the **SetName** option, and type your desired name of the gate in the pop-up window.
6. If you would like to modify your gate, you can do so by left clicking on one of the gate points and moving it around.

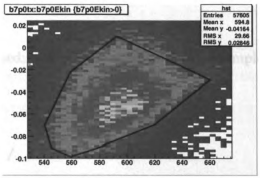
Figure B.3 shows an example of a 2d gate created using the procedure outlined above. Unlike in SpecTcl, your 2d gate will not automatically be drawn every time that you re-draw a histogram, but the gate is still stored into memory until you quit the program. To re-draw a gate the command is:

```
mygate->Draw();
```

You will likely find yourself typing the same gate conditions over and over, which can get tedious. You can store these logical statements in a **TCut** object, and when you do so, the gate conditions can be accessed just by invoking the name of the **TCut**. The syntax to create a TCut is explained with an example: if I wanted to create a



(a) Spectrum showing a parameter with the gate drawn in panel (b) applied to it.



(b) Example of a two dimensional gate. The events outlined in black are included within the gate.

Figure B.3: Example 2d Gate (right panel) and histogram with the gate applied to it (left panel).

`TCut`, named `mycut`, which requires that `b7p0Ekin` and `b13p1Ekin` both be greater than zero, I would type the following:

```
TCut mycut = "b7p0Ekin > 0 && b13p1Ekin > 0" ;
```

The `TCut` can then be used like a normal logical statement, e.g.

```
t->Draw("b7p0x",mycut);
```

Note that in the above command there are no `" "` around `mycut`. This is because the needed `" "` are contained in the definition of `mycut`. `TCuts` can also be combined with "ordinary" gates or other `TCuts` as follows:

```
t->Draw("b7p0x",mycut && "b13p1x>0"); //\ combine a TCut with a "normal" gate.
t->Draw("b7p0x",mycut && mycut2); //\ combine two TCuts.
```

B.7 Creating Pseudo Parameters

Quite often one wants to look at mathematical combinations of the parameters stored in a ROOT file. The simplest way to do this is to simply include the mathematical statements in the argument of the `Draw()` command. For example, if I want to look at the velocity of a particle, which for the purposes of this example is equal to $\sqrt{2 \cdot \texttt{b7p0Ekin}/10}$, then I would type the following:

```
t->Draw("sqrt(2 * b7p0Ekin / 10)");
```

Any valid C++ math operator can be used; some of the more common examples are listed below

- add, subtract, multiply, divide: `+, -, *, /`
- square root of x: `sqrt(x)`
- raise x to the power n: `pow(x,n)`
- sin, cos, tan of x: `sin(x), cos(x), tan(x)`
- arcsin, arccos, arctan of x: `asin(x), acos(x), atan(x)`
- absolute value of x: `abs(x)`

As in the case of gates, you often do not want to type long mathematical expressions over and over again. Fortunately, ROOT allows you to store your mathematical expression in an object called an "alias." For example, if I want to store the velocity expression shown above as an alias called `b7p0vel`, I would do:

```
t->SetAlias("b7p0vel","sqrt(2 * b7p0Ekin / 10)");
```

Now to draw the velocity I just have to type:

```
t->Draw("b7p0vel")
```

Aliases can also be included in the definition of other aliases; for example, if I want to create an alias of 2*velocity, called `b7p0velx2`, then I could do:

```
t->SetAlias("b7p0velx2","2*b7p0vel");
```

Aliases can be used just like any "real" ROOT parameter, e.g. you can draw them, place gates on them (1d and 2d), and so on.

B.8 Using Macros

If you haven't noticed by now, using ROOT involves a lot of typing of commands, and you probably don't want to repeat a bunch of commands over and over again. ROOT

allows you to save series of commands in a macro file; invoking a macro file has the same effect as typing each one of the commands on the command line individually. ROOT macro files should always have the extension `.C`, and all of the commands must be enclosed in a set of curly brackets. The command to run a macro (called e.g. `mymacro.C`) is:

```
.x mymacro.C
```

As a simple example consider the following macro, called `examplemacro.C`; it is located in the `/projects/proj1/MoNA/ROOT/` directory, and the code is reproduced here. Comments in the code are set off by `//` (the comment symbol in C++), and should explain to you what each line is doing.

```
{

//Macro File examplemacro.C

TFile *_file0 = new TFile("/projects/proj1/MoNA/ROOT/example1.C");
//Load the file example1.C into memory

TCut posenergy = "b7p0Ekin > 0 && b13p1Ekin > 0" ;
//Create a TCut

t->Draw("b7p0tx>>hst(100,-.1,.1)",posenergy);
//draw a histogram of b7p0tx, subject to the conditions of posenergy

}
```

You can also invoke other macros from within a macro; for example, if I want to call a macrofile called `macro2.C` from within another macro, I would add the line:

```
gROOT->ProcessLine(".x macro2.C");
```

You can also load in a macro file when you start up ROOT, by typing the name of the macro after the `root.exe` command. For example, if I want to automatically load the file `examplemacro.C` at startup, then the syntax is:

```
> root.exe .x examplemacro.C
```

There is a useful "startup" macro file located in `/projects/proj1/MoNA/ROOT/`; it is called `root_logon.C` It does a few formatting tasks like setting to default color palette to the nice looking RGB one. It also loads up a custom function that allows you to write out a histogram to a two column ASCII file. If you have loaded up `root_logon.C`, then the syntax to write a histogram to file is:

```
hstname->WriteSpec("filename.txt",xlo, xhi);
```

where `xlo` and `xhi` are lower and upper limits of the histogram.

B.9 Various Odds and Ends

This section contains a brief overview of some more useful commands that don't really fit anywhere else. They are listed with a brief description of what the command does followed by the command syntax.

- Draw a histogram using data points:

```
t->Draw("7p0x>>hst(100,-.1,.1)","","p");
```

This would draw `hst` with the default point size, which is very small; to change it do:

```
hst->SetMarkerStyle(20); //sets a reasonable point size
hst->Draw("p"); // redraws "hst" using points
```

- Draw a histogram with error bars (calculated as the square root of the number of counts):

```
t->Draw("7p0x>>hstnew(100,-.1,.1)","","e"); //new histogram
hstold->Draw("e"); //existing histogram
```

- Change the color of the lines or points of a histogram already in memory:

```
hst->SetLineColor(4);   //Lines
hst->SetMarkerColor(4); //Points
```

Colors in ROOT are assigned a number, so here I am setting the histograms to be "color 4" (which is blue). To see a list of all available colors and their corresponding numbers, select **View->Colors** from the **TCanvas** menu.

- Scale a histogram (by the number x):

```
hst->Scale(x);
```

- Draw an existing histogram on the same canvas, without erasing the one already there:

```
hst->Draw("same");
```

- You can also combine arguments like **p** and **same**, e.g.

```
hst->Draw("psame"); //Draw hst on the same canvas, using points.
```

- Clear your current canvas:

```
c1->Clear();
```

- Write 2D gates into a text file:

```
ofstream out;
out.open("my2dgates.txt");
gate1->SavePrimitive(out);
gate2->SavePrimitive(out);
gate3->SavePrimitive(out);
out.close();
```

The text file **my2dgates.txt** will now contain C++ code describing the 2D gates gate1, gate2, and gate3. This code can be copied and pasted into a macro file, to allow the user to load the gates into memory in future ROOT sessions.

• Convert a SpecTcl filter file into a ROOT file:

–Run the executable `flt2root.sh` located in `/projects/proj1/MoNA/ROOT/`

This will prompt you for the name of the input filter file and the name of the output root file. It must be run from a 32-bit `spice` machine. The new root file will contain a `TTree` called `h1` that contains all of the parameters written in the filter file. Any dots in the parameter names will be converted into underscores (e.g. `sweeper.fp.crdc1.x` becomes `sweeper_fp_crdc1_x`), and any uppercase letters past the first letter of the name will be made lowercase (`ToF_hit_1` becomes `Tof_hit_1`). Also, since ROOT needs to assign a value to every parameter for every event, any "not valid" events in the filter file will be given a value of exactly zero in the ROOT file. Finally, there is a limitation on both the size of the filter files and the number of parameters allowed per file, which the author of this manual has not yet determined. This limit is fairly large, however, and can be worked around by making judicious use of Friend Trees and `TChains`.

If you are interested in using the more advanced functionality of ROOT, there are many resources on the web. A good place to start is the official ROOT manual located at:

`http://root.cern.ch`

There is also an online bulletin board ("Root Talk") dedicated to helping users learn to use ROOT located at:

`http://root.cern.ch/phpBB2/`

Bibliography

[1] K. Krane, Introductory Nuclear Physics, Wiley, New York, 1988.

[2] N. Frank, A. Schiller, T. Baumann, D. Bazin, J. Brown, P. DeYoung, J. E. Finck, A. Gade, J. Hinnefeld, R. Howes, J. L. Lecouey, B. Luther, W. Peters, H. Scheit, M. Thoennessen, Observation of the First Excited State in ^{23} O, in: Proceedings of the 16th International Conference on Cyclotrons and their Applications, 2007.

[3] R. Kalpakchieva, H. Bohlen, W. von Oertzen, B. Gebauer, M. von Lucke-Petsch, T. Massey, A. Ostrowski, T. Stolla, M. Wilpert, T. Wilpert, Eur. Phys J. A **7** (2000) 451.

[4] H. G. Bohlen, R. Kalpakchieva, W. von Oertzen1, T. N. Massey, A. A. Ogloblin, G. de Angelis, M. Milin, C. Schulz1, T. Kokalova1, C. Wheldon, Eur. Phys J. A **31** (2007) 279.

[5] A. A. Korsheninnikov, M. S. Golovkov, A. Ozawa, E. A. Kuzmin, E. Y. Nikolskii, K. Yoshida, B. G. Novatskii, A. A. Ogloblin, I. Tanihata, Z. Fulop, K. Kusaka, K. Morimoto, H. Otsu, H. Petrascu, F. Tokanai, Phys. Rev. Lett. **82** (1999) 3581.

[6] A. H. Wuosmaa, K. E. Rehm, J. P. Greene, D. J. Henderson, R. V. F. Janssens, C. L. Jiang, L. Jisonna, E. F. Moore, R. C. Pardo, M. Paul, D. Peterson, S. C. Pieper, G. Savard, J. P. Schiffer, R. E. Segel, S. Sinha, X. Tang, R. B. Wiringa, Phys. Rev. C **72** (2005) 061301(R).

[7] F. Skaza, V. Lapoux, N. Keeley, N. Alamanos, E. C. Pollacco, F. Auger, A. Drouart, A. Gillibert, D. Beaumel, E. Becheva, Y. Blumenfeld, F. Delaunay, L. Giot, K. W. Kemper, L. Nalpas, A. Obertelli, A. Pakou, R. Raabe, P. Roussel-Chomaz, J.-L. Sida, J.-A. Scarpaci, S. Stepantsov, R. Wolski, Phys. Rev. C **73** (2006) 044301.

[8] A. Schiller, N. Frank, T. Baumann, D. Bazin, B. A. Brown, J. Brown, P. A. DeYoung, J. E. Finck, A. Gade, J. Hinnefeld, R. Howes, J.-L. Lecouey, B. Luther, W. A. Peters, H. Scheit, M. Thoennessen, J. A. Tostevin, Phys. Rev. Lett. **99** (2007) 112501.

[9] C. S. Sumithrarachchi, D. J. Morrissey, B. A. Brown, A. D. Davies, D. A. Davies, M. Fancina, E. Kwan, P. F. Mantica, M. Portillo, Y. Shimbara, J. Stoker, R. R. Weerasiri, Phys. Rev. C **75** (2007) 024305.

[10] K. W. Scheller, J. Görres, J. G. Ross, M. Wiescher, R. Harkewicz, D. J. Morrissey, B. M. Sherrill, M. Steiner, N. A. Orr, J. A. Winger, Phys. Rev. C **49** (1994) 46.

[11] Z. Radivojevic, P. Baumann, E. Caurier, J. Cederkall, S. Courtin, P. Dessagne, A. Jokinen, A. Knipper, G. L. Scornet, V. Lyapin, C. Miehe, F. Nowacki, S. Nummela, M. Oinonen, E. Poirier, M. Ramdhane, W. H. Trzaska, G. Walter, J. Aysto, Nucl. Instr. and Meth. A **481** (2002) 464.

[12] M. Zinser, F. Humbert, T. Nilsson, W. Schwab, H. Simon, T. Aumann, M. J. G. Borge, L. V. Chulkov, J. Cub, T. W. Elze, H. Emling, H. Geissel, D. Guillemaud-Mueller, P. G. Hansen, R. Holzmann, H. Irnich, B. Jonson, J. V. Kratz, R. Kulessa, Y. Leifels, H. Lenske, A. Magel, A. C. Mueller, G. Munzenberg, F. Nickel, G. Nyman, A. Richter, K. Riisager, C. Scheidenberger, G. Schrieder, K. Stelzer, J. Stroth, A. Surowiec, O. Tengblad, E. Wajda, E. Zude, Nucl. Phys. A **619** (1997) 151.

[13] S. D. Pain, W. N. Catford, N. A. Orr, J. C. Angelique, N. I. Ashwood, V. Bouchat, N. M. Clarke, N. Curtis, M. Freer, B. R. Fulton, F. Hanappe, M. Labiche, J. L. Lecouey, R. C. Lemmon, D. Mahboub, A. Ninane, G. Normand, N. Soic, L. Stuttge, C. N. Timis, J. A. Tostevin, J. S. Winfield, V. Ziman, Phys. Rev. Lett. **85** (2000) 032502.

[14] F. Deák, A. Horváth, A. Kiss, Z. Seres, A. Galonsky, C. K. Gelbke, H. Hama, L. Heilbronn, D. Krofcheck, W. G. Lynch, D. W. Sackett, H. R. Schelin, M. B. Tsang, J. Kasagi, T. Murakami, Phys. Rev. C **52** (1995) 219.

[15] F. Deak, A. Kiss, Z. Seres, G. Caskey, A. Galonsky, B. Remington, Nucl. Instr. and Meth. A **258** (1987) 67.

[16] L. Heilbronn, A. Galonsky, C. K. Gelbke, W. G. Lynch, T. Murakami, D. Sackett, H. Schelin, M. B. Tsang, F. Deák, A. Kiss, Z. Seres, J. Kasagi, B. A. Remington, Phys. Rev. C **43** (1991) 2318.

[17] O. Tarasov, D. Bazin, Nucl. Instr. and Meth. A **746** (2004) 411.

[18] R. A. Kryger, A. Azhari, A. Galonsky, J. H. Kelley, R. Pfaff, E. Ramakrishnan, D. Sackett, B. M. Sherrill, M. Thoennessen, J. A. Winger, S. Yokoyama, Phys. Rev. C **47** (1993) R2439.

[19] M. Thoennessen, S. Yokoyama, A. Azhari, T. Baumann, J. A. Brown, A. Galonsky, P. G. Hansen, J. H. Kelley, R. A. Kryger, E. Ramakrishnan, P. Thirolf, Phys. Rev. C **59** (1999) 111.

[20] M. Thoennessen, S. Yokoyama, P. Hansen, Phys. Rev. C **63** (2000) 014308.

[21] F. Marti, P. Miller, D. Poe, M. Steiner, J. Stetson, X. Y. Wu, Commissioning of the coupled cyclotron system at nscl, in: Proceedings of the 16th International Conference on Cyclotrons and their Applications, 2001, p.64.

[22] D. J. Morrissey, B. M. Sherrillb, M. Steinerb, A. Stolzb, I. Wiedenhoeverb, Nucl. Instr. and Meth. B **204** (2003) 90.

[23] M. Bird, S. Kenney, J. Toth, H. Weijers, J. DeKamp, M. Thoennessen, A. Zeller, IEEE Trans. Applied Superconductivity **15** (2005) 1252.

[24] B. Luther, T. Baumann, M. Thoennessen, J. Brown, P. DeYoung, J. Finck, J. Hinnefeld, R. Howes, K. Kemper, P. Pancella, G. Peaslee, W. Rogers, S. Tabor, Nucl. Instr. and Meth. A **505** (2003) 33.

[25] T. Baumann, J. Boike, J. Brown, M. Bullinger, J. Bychoswki, S. Clark, K. Daum, P. DeYoung, J. Evans, J. Finck, N. Frank, A. Grant, J. Hinnefeld, G. Hitt, R. Howes, B. Isselhardt, K. Kemper, J. Longacre, Y. Lu, B. Luther, S. Marley, D. McCollum, E. McDonald, U. Onwuemene, P. Pancella, G. Peaslee, W. Peters, M. Rajabali, J. Robertson, W. Rogers, S. Tabor, M. Thoennessen, E. Tryggestad, R. Turner, P. VanWylen, N. Walker, Nucl. Instr. and Meth. A **543** (2005) 517.

[26] K. Makino, M. Berz, Nucl. Instr. and Meth. A **558** (2005) 346.

[27] H. Scheit, Simple Track for MoNA, Technical report, NSCL (2006).

[28] N. Frank, Spectroscopy of Neutron Unbound States in Neutron Rich Oxygen Isotopes, Ph.D. thesis, Michigan State University (2006).

[29] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, F. Rossi, GNU Scientific Library Reference Manual, `http://www.gnu.org/software/gsl/manual/html_node/` (2007).

[30] R. Glauber, Lectures in Theoretical Physics, Vol. I, Interscience, New York, 1959.

[31] A. Schiller, N. Frank, The Problem of Track Reconstruction in the Sweeper Magnet, Technical report, NSCL (2005).

[32] N. Frank, A. Schiller, D. Bazin, W. Peters, M. Thoennessen, Nucl. Instr. and Meth. A **580** (2007) 1478.

[33] D. Tilley, J. Kelley, J. Godwin, D. Millener, J. Purcell, C. Sheu, H. Weller, Nucl. Phys. A **745** (2004) 155.

[34] H. Simon, M. Meister, T. Aumann, M. Borge, L. Chulkov, U. D. Pramanik, T. Elze, H. Emling, C. Forssen, H. Geissel, M. Hellstrom, B. Jonson, J. Kratz, R. Kulessa, Y. Leifels, K. Markenroth, G. Munzenberg, F. Nickel, T. Nilsson, G. Nyman, A. Richter, K. Riisager, C. Scheidenberger, G. Schrieder, O. Tengblad, M. Zhukov, Nucl. Phys. A **791** (2007) 267.

[35] A. Amelin, M. Gornov, Y. Gurov, A. II'in, P. Morokhov, V. Pechkurov, V. Savel'ev, F. Sergeev, S. Smirnov, B. Chernyshev, R. Shafigullin, A. Shishkov, Sov. J. Nucl. Phys. **52** (1990) 782.

[36] B. M. Young, W. Benenson, J. H. Kelley, N. A. Orr, R. Pfaff, B. M. Sherrill, M. Steiner, M. Thoennessen, J. S. Winfield, J. A. Winger, S. J. Yennello, A. Zeller, Phys. Rev. C **49** (1994) 279.

[37] M. Zinser, F. Humbert, T. Nilsson, W. Schwab, T. Blaich, M. J. G. Borge, L. V. Chulkov, H. Eickhoff, T. W. Elze, H. Emling, B. Franzke, H. Freiesleben, H. Geissel, K. Grimm, D. Guillemaud-Mueller, P. G. Hansen, R. Holzmann, H. Irnich, B. Jonson, J. G. Keller, O. Klepper, H. Klingler, J. V. Kratz, R. Kulessa, D. Lambrecht, Y. Leifels, A. Magel, Phys. Rev. Lett. **75** (1995) 1719.

[38] M. G. Gornov, Y. B. Gurov, S. V. Lapushkin, P. V. Morokhov, V. A. Pechkuriv, K. Seth, T. Pedlar, J. Wise, D. Zhao, Bull. Russ. Acad. Sci., Phys. Ser. **62** (1998) 1781.

[39] L. Chen, B. Blank, B. A. Brown, M. Chartier, A. Galonsky, P. G. Hansen, M. Thoennessen, Phys. Lett. B **505** (2001) 21.

[40] M. Chartier, J. R. Beene, B. Blank, L. Chen, A. Galonsky, N. Gan, K. Govaert, P. G. Hansen, J. Kruse, V. Maddalena, M. Thoennessen, R. L. Varner, Phys. Lett. B **510** (2001) 24.

[41] H. Jeppesen, A. Moro, U. Bergmann, M. Borge, J. Cederkall, L. Fraile, H. Fynbo, J. Gomez-Camacho, H. Johansson, B. Jonson, M. Meister, T. Nilsson, G. Nyman, M. Pantea, K. Riisager, A. Richter, G. Schrieder, T. Sieber, O. Tengblad, E. Tengborn, M. Turrion, F. Wenander, Phys. Lett. B **642** (2006) 449.

[42] G. Audi, O. Bersillon, J. Blachot, A. H. Wapstra, Nucl. Phys. A **624** (1997) 1.

[43] F. Ajzenberg-Selove, J. Kelley, Nucl. Phys. A **506** (1990) 1.

[44] W. Peters, Study of Neutron Unbound States Using the Modular Neutron Array (MoNA), Ph.D. thesis, Michigan State University (2007).

[45] P. Descouvemont, Phys. Lett. B **331** (1994) 271.

[46] P. Descouvemont, Phys. Rev. C **52** (1995) 704.

[47] M. Labiche, F. M. Marqués, O. Sorlin, N. V. Mau, Phys. Rev. C **60** (1999) 027303.

[48] J. Lecouey, Few Body Systems **34** (2004) 21.

[49] M. Stanoiu, F. Azaiez, Z. Dombradi, O. Sorlin, B. A. Brown, M. Belleguic, D. Sohler, M. G. S. Laurent, M. J. Lopez-Jimenez, Y. E. Penionzhkevich, G. Sletten, N. L. Achouri, J. C. Angelique, F. Becker, C. Borcea, C. Bourgeois, A. Bracco, J. M. Daugas, Z. Dlouhy, C. Donzaud, J. Duprat, Z. Fulop, D. Guillemaud-Mueller, S. Grevy, F. Ibrahim, A. Kerek, A. Krasznahorkay, M. Lewitowicz, S. Leenhardt, S. Lukyanov, P. Mayet, S. Mandal, H. van der Marel, W. Mittig, J. Mrazek, F. Negoita, F. D. Oliveira-Santos, Z. Podolyak, F. Pougheon, M. G. Porquet, P. Roussel-Chomaz, H. Savajols, Y. Sobolev, C. Stodel, J. Timar, A. Yamamoto, Phys. Rev. C **69** (2004) 034312.

[50] Z. Elekes, Z. Dombrádi, N. Aoi, S. Bishop, Z. Fülöp, J. Gibelin, T. Gomi, Y. Hashimoto, N. Imai, N. Iwasa, H. Iwasaki, G. Kalinka, Y. Kondo, A. A. Korsheninnikov, K. Kurita, M. Kurokawa, N. Matsui, T. Motobayashi, T. Nakamura, T. Nakao, E. Y. Nikolskii, T. K. Ohnishi, T. Okumura, S. Ota, A. Perera, A. Saito, H. Sakurai, Y. Satou, D. Sohler, T. Sumikama, D. Suzuki, M. Suzuki, H. Takeda, S. Takeuchi, Y. Togano, Y. Yanagisawa, Phys. Rev. Lett. **98** (2007) 102502.

[51] D. Bazin, et al., Nucl. Instr. and Meth. A **204** (2003) 629.

[52] D. Geesaman, C. Gelbke, R. Janssens, B. Sherrill, Ann. Rev. Nucl. Part. Sci. **56** (2006) 53.