DELAYED NEUTRON EMISSION MEASUREMENTS FOR U-235 AND Pu-239

A Thesis

by

YONG CHEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2006

Major Subject: Health Physics

DELAYED NEUTRON EMISSION MEASUREMENTS FOR U-235 AND Pu-239

A Thesis

by

YONG CHEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Warren D. Reece
Committee Members,	Rand L. Watson
	William S. Charlton
Head of Department,	William E. Burchill

December 2006

Major Subject: Health Physics

ABSTRACT

Delayed Neutron Emission Measurements for U-235 and Pu-239.

(December 2006)

Yong Chen, B.S., Tsinghua University;

M.S., Tsinghua University

Chair of Advisory Committee: Dr. W. Dan Reece

The delayed neutron emission rates of U-235 and Pu-239 samples were measured accurately from a thermal fission reaction. A Monte Carlo calculation using the Geant4 code was used to demonstrate the neutron energy independence of the detector used in the counting station.

A set of highly purified actinide samples (U-235 and Pu-239) was irradiated in these experiments by using the Texas A&M University Nuclear Science Center Reactor. A fast pneumatic transfer system, an integrated computer control system, and a graphite-moderated counting system were constructed to perform all these experiments. The calculated values for the five-group U-235 delayed neutron parameters and the six-group Pu-239 delayed neutron parameters were compared with the values recommended by Keepin et al. (1957) and Waldo et al. (1981). These new values differ slightly from literature values. The graphite-moderated counting station and the computerized pneumatic system are now operational for further delayed neutron measurement.

ACKNOWLEDGMENTS

I want to thank my committee chairman, Dr. W. Dan Reece, for his orientation, guidance, and support toward the completion of this project. I want to also thank my committee members, Dr. Leslie A. Braby, Dr. William S. Charlton, Dr. John W. Poston, Sr. and Dr. Rand L. Watson, for their time and effort dedicated to this thesis. I would like to express my special appreciation to the Texas A&M University Nuclear Science Center Staff (especially Tom Fisher, Alfred Hanna, Chad Everett, and Joe Snook) for their patience and cooperation through the design, development, and construction of this project.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

A prompt neutron is a neutron immediately emitted by a nuclear fission event (10^{-14} s), as opposed to a delayed neutron which is emitted by one of the fission products anytime from a few milliseconds to a few minutes later. These "delayed neutrons" are the result of the beta transitions that occur after fission product decay (Roberts et al. 1939).

There are over 270 radionuclides that have been identified as delayed neutrons precursors (DNP) (Brady and England 1989). Keepin et al. (1957) reported delayed neutron emission data by using six groups. Twelve parameters are needed to define a set of the six-group delayed neutron data for a specific fissile nuclide and specific fission-inducing neutron energy. These parameters include six relative yields (known as A), and six decay constants (known as λ) and I is the pseudo group number. The equation for delayed neutron emission rate (DNP(t)) is shown in Eq. (1).

$$DNP(t) = \sum_{i=1}^{I} A_i e^{-\lambda_i t} . \quad (1)$$

Although this is a very mature field, we have found many interesting things while investigating delayed neutron parameters. Reece and Wang (2005) used Monte Carlo calculations to perform sensitivity studies on the uncertainty of the individual delayed neutron constants. They found that the system of variables used to describe delayed neutron yields constitutes an ill-posed problem, meaning that arbitrarily small changes in

---

This thesis follows the style of Health Physics.

the data will produce arbitrarily large changes in the constants themselves. This suggests that small errors in flight times and the different energy responses of the detectors for individual groups could have a large effect on the assignment of delayed neutron parameters.

1.1 OBJECTIVE

The primary objective of this work was to use the Texas A&M University Nuclear Science Center Reactor (NSCR) and a special designed graphite-moderated counting system to measure the time-dependent delayed neutron emission rates. We irradiated samples in the NSCR at a position with highly thermalized neutron fluence rate. The sample was pneumatically transferred to a counting station composed of graphite and ^{3}He detectors. The geometry was optimized to minimize any energy dependence in the energy spectrum of the delayed neutrons. Much effort was put into accurately assessing the flight time of the sample as it leaves the reactor until it enters the counting station. An in-core switch and an optical sensor in the detector array were used to get precise sample flight times. Three multi-channel analyzers (MCA), using in multi-scaler mode were used to record the detector signals. We developed a computer program to control the pneumatic sample transfer system. Three samples (two U-235s and one Pu-239) were irradiated and measured in this project.

1.2 THEORY

The mechanism of delayed neutron emission in fission is well understood in principle (Bohr and Wheeler 1939). The beta-decay of a nuclide (Z, N) with high decay energy, usually called the delayed neutron precursor (DNP) can populate excited states. The daughter nuclide (Z+ l, N-1) in excited state may be lying above the neutron binding energy. The daughter nuclide can possibly de-excite into the nucleus (Z+ 1, N-2) through the emission of a neutron. The timescale of these emissions of delayed neutrons from the daughter nuclide is controlled by the half-life of the parent nuclide (Z, N). Those radionuclides which have a few neutrons in excess of a closed neutron shell are most likely to emit delayed neutron through this process.

Fig. 1 is a typical decay scheme with delayed neutron emission for precursor Br-87 (Charlton 1998). Br-87 decays to ground state Kr-87 and excited state Kr-87* by beta emission with a 55 s half life. Then Kr-87* can decay by neutron emission. The half life of this neutron emission depends on the half life of Br-87's beta emission.

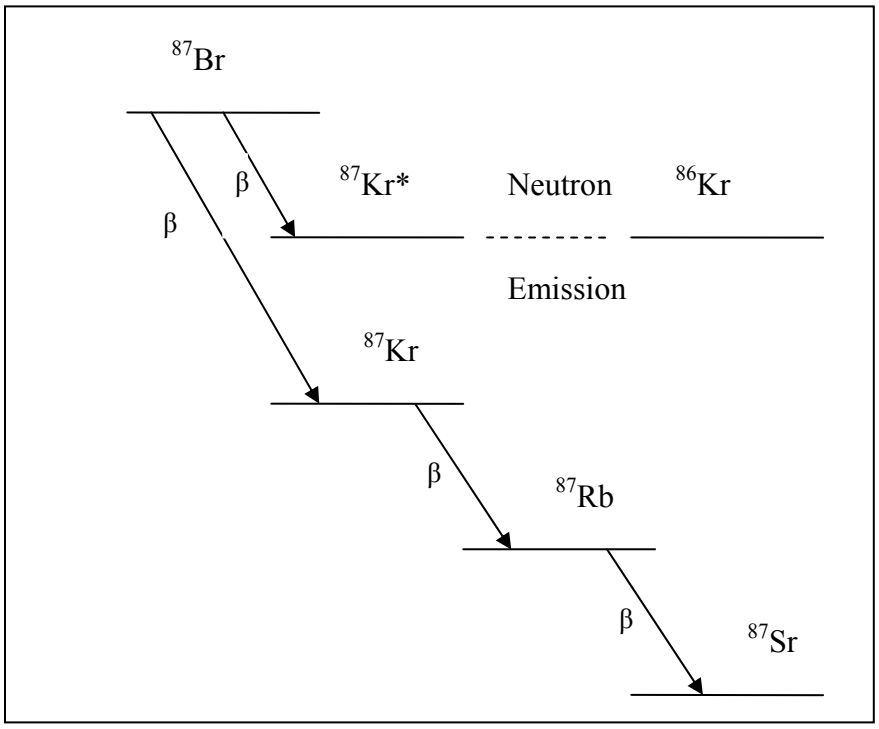


Fig. 1. Decay scheme for Br-87 (with 55 s half- life).

The production rate of a particular delayed neutron precursor (DNP) during irradiation is determined below.

$$\frac{dN}{dt} = Y_C \sum_f \phi - \lambda N \ , \tag{2}$$

where N is the atom density of the DNP, Y_C is the yield probability of the DNP from a fission reaction, $\Sigma_f\Phi$ is the fission rate and λ is the decay constant of the DNP. The neutron capture cross section of the DNP is ignored in this equation and the parents of the DNP are assumed to decay instantly after the fission reaction.

This equation can easily be solved for N as function of the irradiation time, t, as shown in Eq. (3).

$$N\ (t) = \frac{Y_C \Sigma_f \phi}{\lambda}(1 - e^{-\lambda t}) . \quad (3)$$

To obtain the neutron emission rate from Eq. (2), we multiply both sides by the decay constant of the DNP and the probability that the DNP undergoes decay by the neutron emission (Pn). Further, the neutron emission rate after an actinide sample has been removed from the reactor can be found by knowing the irradiation time, t. The equation for the delayed neutron emission rate [DNP(t)], from all of the DNPs (the total number of DNP is, i, Pn_i, Yc_i and λ_i represents for a special DNP)at time, t, after an irradiation for a time, t_r, is:

$$DNP(t) = \sum_{i=1}^{I} Pn_i Yc_i \Sigma_f \phi (1 - e^{-\lambda_i t_r}) e^{-\lambda_i t} . \quad (4)$$

We choose t_r (about 200 s) for detection time that is much larger than the half-life of the longest lived DNP [the longest half-life of DNP (Br-87) is 55.6 s], then Eq. (4) reduces to:

$$DNP(t) = \sum_{i=1}^{I} Pn_i Yc_i \Sigma_f \phi e^{-\lambda_i t} \quad (5)$$

To organize the large number of DNPs, Keepin et al. (1957) introduced the suggestion of the "six-group" pseudo model to describe delayed neutron emission for both fast and thermal neutron-induced fission. The relative yields assigned to six-group based model [shown in Eq. (7)] can be determined from the value of v_D (the total number of delayed neutrons emitted per fission) which comes from the delayed neutron emission probabilities and cumulative yields for each precursor in this pseudo group. The relative yield due to several DNP's in a pseudo group is shown in Eq. (7) (N represents the number

of DNPs in the special pseudo group).

$$\nu_{Dj} = \sum_{i=1}^{N} Pn_i Yc_i \ , \tag{6}$$

$$\alpha_j = \frac{\upsilon_{Dj}}{\sum_{j}^{6} \upsilon_{Dj}} \ . \tag{7}$$

Using Eq. (6) and Eq. (7), Eq. (5) can be simplified to the result shown in Eq. (8).

$$DNP(t) = \nu_{Dj} \Sigma_f \phi \sum_{j=1}^{6} \alpha_j e^{-\lambda_j t} \ . \tag{8}$$

We combine all the unknown quantities on the right hand side of Eq. (8) are combined and a new variable A_i is defined as shown in Eq. (9)

$$DNP(t) = \sum_{j=1}^{6} A_j e^{-\lambda_j t} \ . \tag{9}$$

This six pseudo group model was used in this thesis for the calculation of delayed neutron parameters. Table 1 shows the literature values of the delayed neutron parameters for U-235 irradiated by thermal neutrons (Keepin et al. 1957).

Table 1. Six-group model with twelve parameters for U-235.

Group	DNP	Half-life (s)	Yield
1	^{87}Br	55.9	0.033
2	^{88}Br	22.7	0.219
3	^{93}Rb	6.24	0.196
4	^{139}I	2.3	0.395
5	^{91}Br	0.61	0.115
6	^{96}Rb	0.23	0.042

1.3 THE HISTORY OF DELAYED NEUTRON MEASUREMENT SYSTEMS

The delayed neutron measurement systems always consist of three major components: (1) a neutron source for sample irradiation, (2) a sample transfer system, and (3) a detector assembly counting system. The neutron source may be a sample irradiated in nuclear reactor, a neutron generator, or a spontaneous fission source, such as AmBe. Most neutron sources [(α, n) or (γ, n) reaction] have the higher neutron average energy than that of the delayed neutron DNPs. The neutron source may be moderated by lower atom number materials (like hydrogen or carbon). For the second part, most systems use a pneumatic transfer technique. The samples were sent to and from the irradiation position in a sealed capsule. The primary requirements of the sample transfer system are speed and reliability.

The third component of the DN measurement system consists of a detector system and the associated electronics. The detector system includes the neutron detectors, a moderator and shielding station. The most common neutron detectors have been used is BF_3 detectors because of their good sensitivity and reasonable low cost. The more efficient ^3He neutron detectors have been used as well though more expensive. A disadvantage of ^3He detectors is their greater relative sensitivity to gamma radiation. Since delayed neutrons have a wide range energy spectrum, the energy efficiencies are extremely important to each measurement system.

Keepin et al. (1957) performed a detailed study of delayed neutron emission at the Los Alamos National Laboratory using Godiva Reactor between the years 1954-1975. The “six-group” concept introduced as his major results was widely used. The neutron

detector was a 1.25 cm diameter BF_3 proportional counter in "long" geometry (Hanson and Mckibben 1947), modified by adding a shaped sleeve of boron plastic around the central BF_3 tube. The flight time of the fissile sample from the point of irradiation to the counting station was 50 milliseconds.

Jewell et al. (1968) at the Lawrence Radiation Laboratory were the first to study an energy-independent, high neutron-efficiency graphite-moderated counting system. They used 40 1.5 m long BF_3 proportional counters imbedded in a graphite cylinder (1.83 m long and 1.53 m in diameter) surrounded by 60 cm water shielding for absorbing and reflecting neutrons.

Waldo et al. (1981) used the Lawrence Livermore Pool-Type Reactor for thermal fission delayed neutron measurements. The counting station consisted of 20 ^{3}He detectors placed concentrically around the sample and embedded in polyethylene. The rabbit transfer time was 1 s and the detector dead time was 3.1 μs. To decrease the sample transfer time, one detector was placed just above the reactor pool to get the shortest sample transfer distance. Because of the long sample flight time, he introduces a "five-group" model.

Charlton (1998) used NSCR for fast fission delayed neutron measurements. The detector array consisted with three BF_3 tubes embedded in a 40 cm polyethylene cylindrical block. A cadmium sheath surrounded the outside of the block to absorb any extraneous or background source of thermal neutrons.

1.4 PREVIOUS EFFORTS AT UNCERTAINTY FOR DELAYED NEUTRON PARAMETERS

As mentioned previously, Reece and Wang (2005) are exploring the methodology of assigning values to delayed neutron parameters. Three FORTRAN programs were written and used to simulate the measurement of delayed neutrons. The literature values for the six delayed neutron groups were used to generate the delayed neutron count rate as a function of time. The time steps were chosen to simulate the dwell time in a MCA. The count in a particular channel is simply the neutron count rate times the dwell time. The initial time steps were in 25-millisecond increments up to 10 s, and then 0.5-s time steps are taken up to 100 s. Finally, 10-s time steps are taken up to 280 s.

There are experimental and conceptual limits on how small or large the time steps can be: too long and the count rate changes significantly during the time step; too short and the Poisson variation of the counts is increased excessively. Based on a few calculations in which the dwell times were adjusted, these time steps were found to be close to optimal. Time steps shorter than about 25 milliseconds are difficult to use for a variety of reasons, but fortunately 25 milliseconds is sufficiently short so that the shortest-lived group can still be adequately quantified. The breaks at 10 seconds and 100 seconds are because the counts in a channel are nearing 2500, the limit for 2% relative uncertainty for Poisson distributed variables, and because the length of the longer time step will have less than 5%-10% change during the new dwell time.

After assigning time steps and computing the counts at each time, the FORTRAN programs take the counts from a particular time channel based on theoretical count rate

and dwell time. These counts are distributed randomly using the IMSL (Visual Numerics 2005) ANORIN routine that uses an inverse Poisson distribution.

These randomly distributed counts are used as "simulated data" in each of the three codes to find the six-group yields, the six decay constants, and an arbitrary constant that includes the fluence, sample size, and detector efficiency. The three algorithms are used to search for those variables that minimize the sum of the square of the differences between the simulated data and the fits generated using the algorithms.

1.4.1 Algorithm 1 matrix inversion

These three codes diverge in their methods to estimate the original parameters used to generate the simulated data. The matrix inversion method is the algorithm used by Keepin et al. (1957), Waldo et al. (1981), and others to optimize the selection of group parameters.

The delayed neutron counts, in a MCA, are governed by Eq. (10).

$$Y(t) = \sum_{i=1}^{I} p(t) A_i^0 e^{-\lambda_i^0 t} , \tag{10}$$

where A_i is the yield of the i^{th} precursor group, λ_i is the decay constant for the i^{th} group, I is the number of groups (assumed to be six in this thesis), p is a proportionality constant that depends on the neutron fluence rate, the mass and isotopic purity of the sample, the detector efficiency, and the dwell time of the MCA. It is also important that the irradiation times are long enough so that the delayed neutron precursors are at saturation. The superscript 0 designates that these are the optimal values. When fitting experimental data,

the difference between observed data and the best guessed A and λ values can be written as Eq. (11):

$$Z(t) = Y(t) - \sum_{i=1}^{I} p(t) A_i e^{-\lambda_i t} , \tag{11}$$

where with no superscript 0, the A's and λ's are estimates of A^0 and λ^0, and Z is the difference between observed and estimated data. The guess of A and λ can be improved by substituting:

$$\begin{aligned} A_i^{new} &= A_i + \Delta A_i \\ \lambda_i^{new} &= \lambda_i + \Delta \lambda_i \end{aligned} . \tag{12}$$

If E^2 is the square of the differences between the experimental and calculated points, it is also the squared differences between Z(t) and the contributions of the ΔA_i's and $\Delta\lambda_i$'s, giving:

$$E^2 = \sum_{i=0}^{\infty} W(t) \left[Z(t) - \sum_{i=0}^{I} p(t) e^{-\lambda_i t} (\Delta A_i - A_i \Delta \lambda_i t) \right]^2 , \tag{13}$$

where W (t) is the reciprocal of the variance of the data point at t. When E^2 is minimized, this is the best fit of the data. At this minimum, the derivatives will approach zero. If

$$\begin{aligned} H(L,i) &= \sum_{i=0}^{\infty} W(t) \left[p(t) e^{-\lambda_L t} e^{-\lambda_i t} \right] \\ D(L) &= \sum_{i=0}^{\infty} W(t)\, p(t) e^{-\lambda_L t} Z(t) \end{aligned} , \tag{14}$$

$$\begin{aligned} F(L) &= \sum_{i=0}^{\infty} W(t) e^{-\lambda_L t} Z(t) t \\ G(L,i) &= \sum_{i=0}^{\infty} W(t)\, p(t) t^2 e^{-\lambda_L t} e^{-\lambda_i t} (-A_i) \end{aligned} , \tag{15}$$

ΔA_i and $\Delta\lambda_i$ can be found by inverting the G and H matrix and solving for the individual ΔA_i and $\Delta\lambda_i$ terms like so:

$$\begin{pmatrix} \vdots \\ \Delta A_i \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots & \\ \cdots & H(L,i) \quad \cdots \\ \vdots & \end{pmatrix}^{-1} \begin{pmatrix} \vdots \\ D(L) \\ \vdots \end{pmatrix}, \tag{16}$$

$$\begin{pmatrix} \vdots \\ \Delta\lambda_i \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots & \\ \cdots & G(L,i) \quad \cdots \\ \vdots & \end{pmatrix}^{-1} \begin{pmatrix} \vdots \\ F(L) \\ \vdots \end{pmatrix}. \tag{17}$$

These new estimates are used in Eq. (12) and the process is continued until the differences between successive iterations are small.

1.4.2 Algorithm 2 Levenberg-Marquardt method

The second method was used by researchers from the late 1960's until today. The heart of the algorithm is based on work by Levenberg (1944) and further developed by Marquardt (1963). Both algorithm 2 and 3 discussed below are simplex algorithms. Simplex algorithms are a class of algorithms that seek maxima or minima by assessing the local gradient among the variables to be optimized and following this gradient until it approaches zero – the location of local maxima or minima.

1.4.3 Algorithm 3 Quasi-Newton method

The last method was from a routine provided by the IMSL library. This routine used a Quasi-Newton method (Gill and Murray 1972) with a finite difference gradient to help locate a minimum. This routine has the advantage of looking over a wide range of

variables to find global minimums.

1.4.4 Results from the three algorithms

This simulation was as close as one can hope to get to a perfect experiment. The flight time of the sample from the reactor after irradiation was zero. The detector dead time for the high count rates in the experiment is zero. There is no energy dependence among the detectors. There are no background counts. There is no drift in voltage or change in sensitivity during the experiment. In short, the only uncertainly in the simulated experiment comes from the Poisson distribution of the counts themselves and the rounding from real numbers to integers to simulate counts. Even these effects are minimized by having a high initial count rate (400,000 cps) and optimized dwell time.

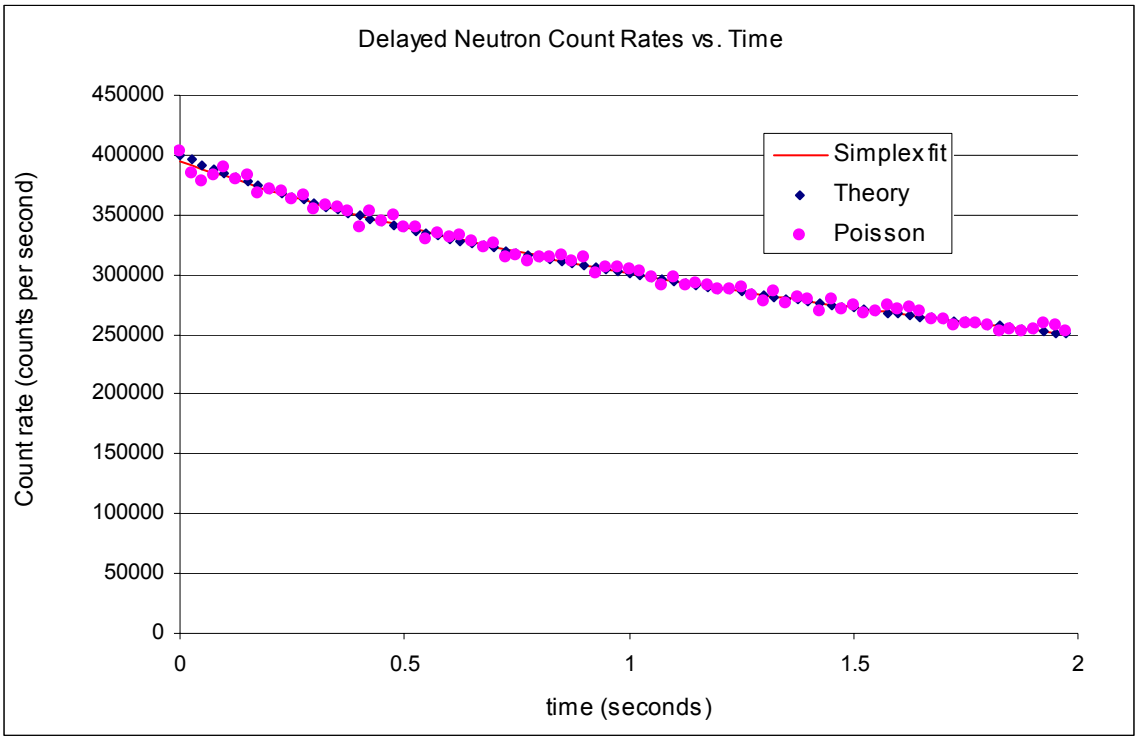


Fig. 2. Delayed neutron count rates vs. time.

Graphs of the theoretical curve and Poisson distributed data and smooth fitted function data are given in Fig. 2.

A glance at the differences between the Poisson distributed values and the values generated from the variables produced by the simplex fit shows how well the values fit the data. Within the resolution of the graphs, the theoretical and simplex fits values can't be distinguished most of the time. A plot of weighted residuals, that is, the deviations divided by the respective weighting factor is shown in Fig. 3. This is a powerful tool for locating bias in a statistical fit. Fig. 3 shows no pattern to the weighted residuals.

Fig. 3. Weighted residuals as a function of time.

Although these fits look superb, there is reason to challenge these results. Table 2 shows the variation of the twelve parameters within different fitting method. These

parameters vary by as much as 73%.

Table 2. Variance of the parameters when using different fitting methods.

	Keepin's matrix	Levenberg-Marquardt	Quasi-Newton
λ_1/λ_1^0	0.9995	1.0005	0.9688
λ_2/λ_2^0	1.0002	1.0001	0.9655
λ_3/λ_3^0	0.9996	0.9992	0.7423
λ_4/λ_4^0	1.0001	0.9988	0.6348
λ_5/λ_5^0	1.0041	1.0310	0.3353
λ_6/λ_6^0	0.9925	0.9917	0.5530
a_1/a_1^0	0.9995	0.9993	0.6726
a_2/a_2^0	1.0004	1.0002	0.7254
a_3/a_3^0	0.9986	1.0002	0.4294
a_4/a_4^0	1.0013	0.9993	0.4865
a_5/a_5^0	0.9908	1.0004	1.7365
a_6/a_6^0	1.0166	1.0398	2.3556
E^2	645.2630	644.5337	640.1092

Even in this ideal measurement, very small changes can have large effects on the delayed neutron parameters. We realized that the different neutron energy for the individual delayed neutron group and the energy responses of the detector array could also have a large effect on the parameters.

The delayed neutron energy spectrum for thermal neutron fission of U-235 with all

the six groups is shown in Fig. 4 (Charlton 1998).

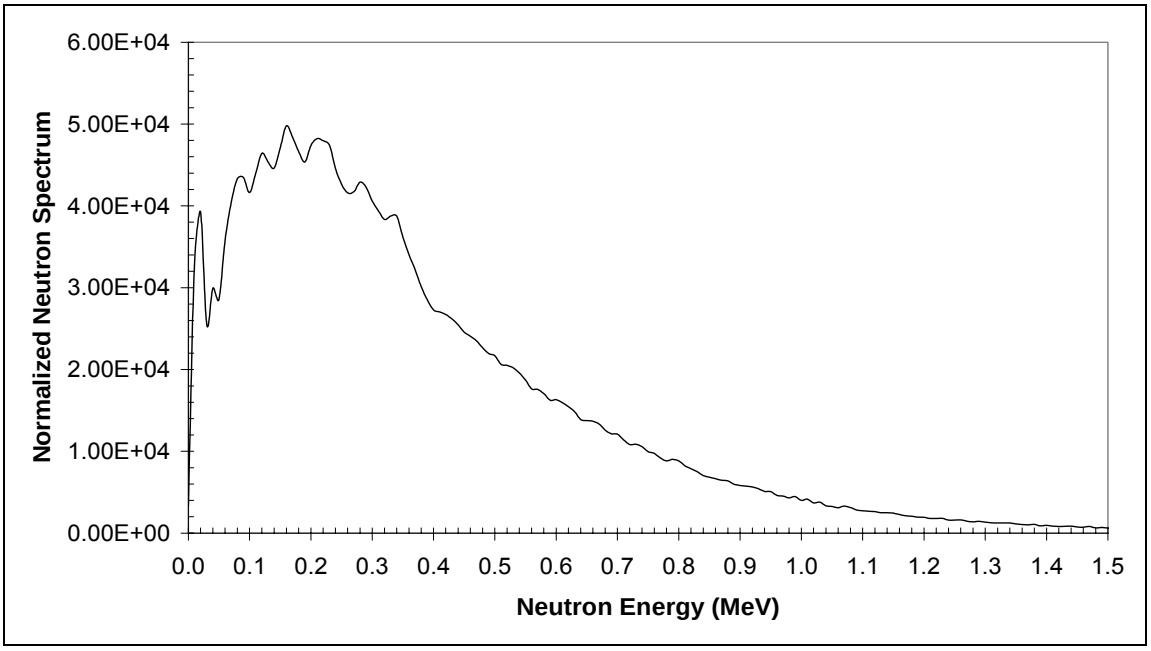


Fig. 4. U-235 delayed neutron energy spectrum.

Table 3. Delayed neutron average energy for U-235 with six-group model.

Delayed neutron Group Number	Average Energy (keV)
1	400.3
2	466.5
3	437.6
4	552.4
5	513.2
6	535.2

The U-235 delayed neutron-energy varies from 0 to 1.5 MeV. From Table 3, we can see the delayed neutron average energy also varies among groups (Brady and England

1989). The size of the neutron detector, moderator material, and counting geometry can make appreciable differences in the detector energy response. As discussed before, this small error in the measurement can result in a large error in the value of the delayed neutron parameters. A neutron-energy independent counting station is needed for our experiment. The new counting station constructed using graphite was designed by using modern Monte Carlo software to minimize the effect of neutron energy.

CHAPTER II

EXPERIMENT SETUP

Pure samples of fissile materials were irradiated in a highly-thermalized position in the NSCR, pneumatically transferred to a graphite-moderated counting system immediately after irradiation and the neutron emission rate counted as a function of time.

The delayed neutron measurements were performed at NSC reactor during last ten years (Saleh 1995, Charlton 1998). The main part of this experiment is a new graphite-moderated counting station. The fast pneumatic transfer system we used follows the design by Charlton (1998). The samples were transferred in the core and irradiated for a pre-selected time period and then were transferred again to a graphite-moderated counting station. The count rates were acquired by three MCAs in a pre-selected dwell time. When the detection was finished, the samples were transferred to a remote storage for further decay. The pneumatic transfer system was controlled by C program written by Yong Chen and Alfred Hanna.

Three ^{3}He detectors (CANBERRA model 2006) were placed at different distances from the counting station receiver. The energy response function of each detector was determined using the GEANT4 code developed by CERN (European Organization for Nuclear Research 2005). A C language program written by Yong Chen and Alfred Hanna was used to drive a CTR05 counter board to control the pneumatic system, including a precise timer to measure flight time with error on the order of 0.001 second.

2.1 PNEUMATIC TRANSFER SYSTEM

The transfer system follows the design used by Charlton (1998). Polyethylene tubing (2.5 cm O.D.) was used for sample line and 2 cm O.D. air hose for CO_2. CO_2 gas was supplied at 80 psi (3 s for each time) to transfer the sample which was controlled by the integrated computer system. A photosensor located at the counting station and a switch sensor in the core was used to measure the sample transit time from the core to the detector. The measured sample transfer time for this system is from 0.6 s to 1.2 s.

An in-core receiver with a switch sensor was designed, built for the irradiation of the samples and accurate flight time measurement. The device consisted of three individually machined aluminum parts with threaded connections. A radiation-resist sealant was used to seal the joints. This in-core receiver can be placed in a pre-installed aluminum stander in the core. 30 feet aluminum gas line and sample line was weld in the receiver to match the distance from the core to the reactor pool water level. A wire fastened inside the gas line was ready for the signal transfer for the switch sensor. Fig. 5 is a cross-sectional view of the internals of the in-core receiver.

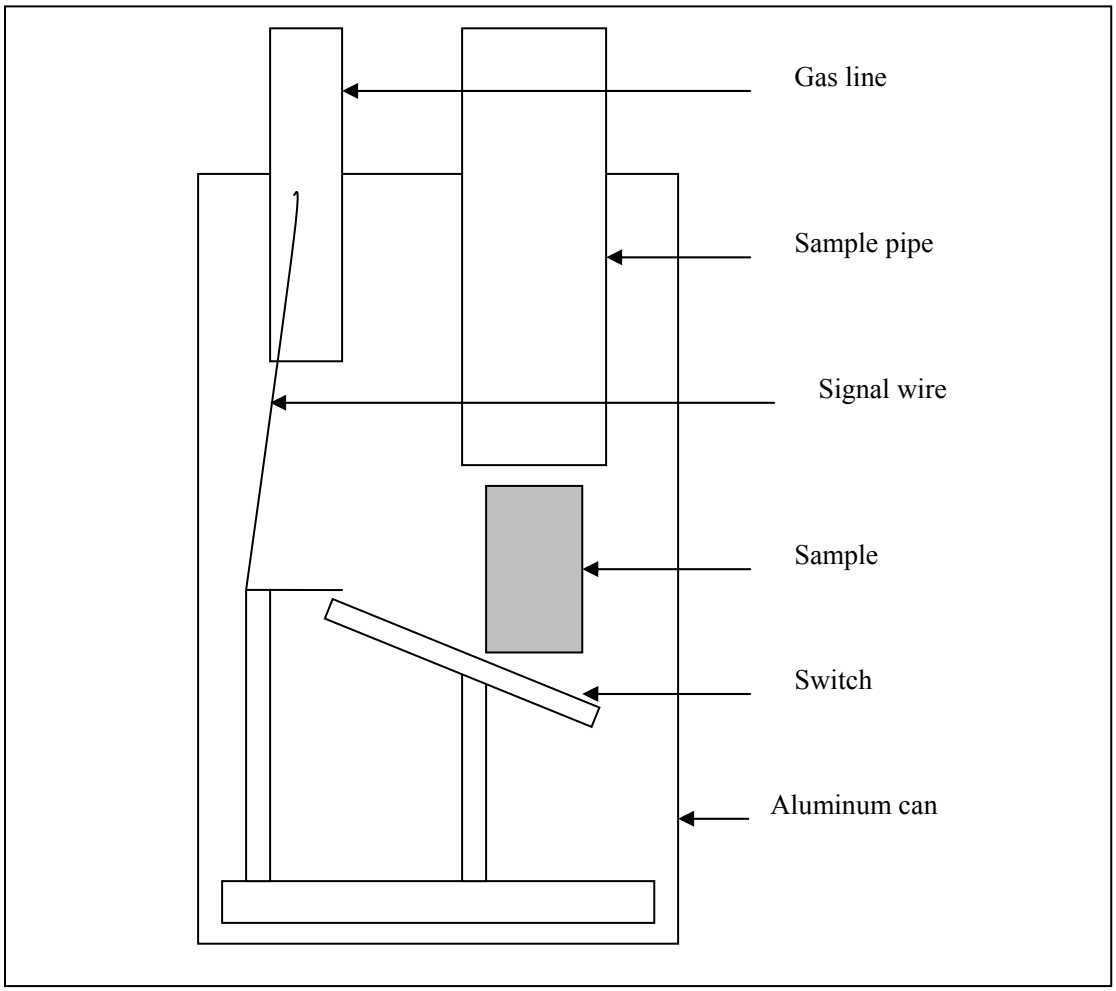


Fig. 5. Pneumatic in-core receiver (cross-sectional view).

When the sample comes to the core receiver, the weight of the sample closes the switch to send out a signal. When the sample leaves the receiver, the different mass of the two sides of the switch makes the switch open. The signals were send out when the sample arrived and leaved the switch through the wire in the gas line.

Fig. 6 shows the configuration of the pneumatic transfer system.

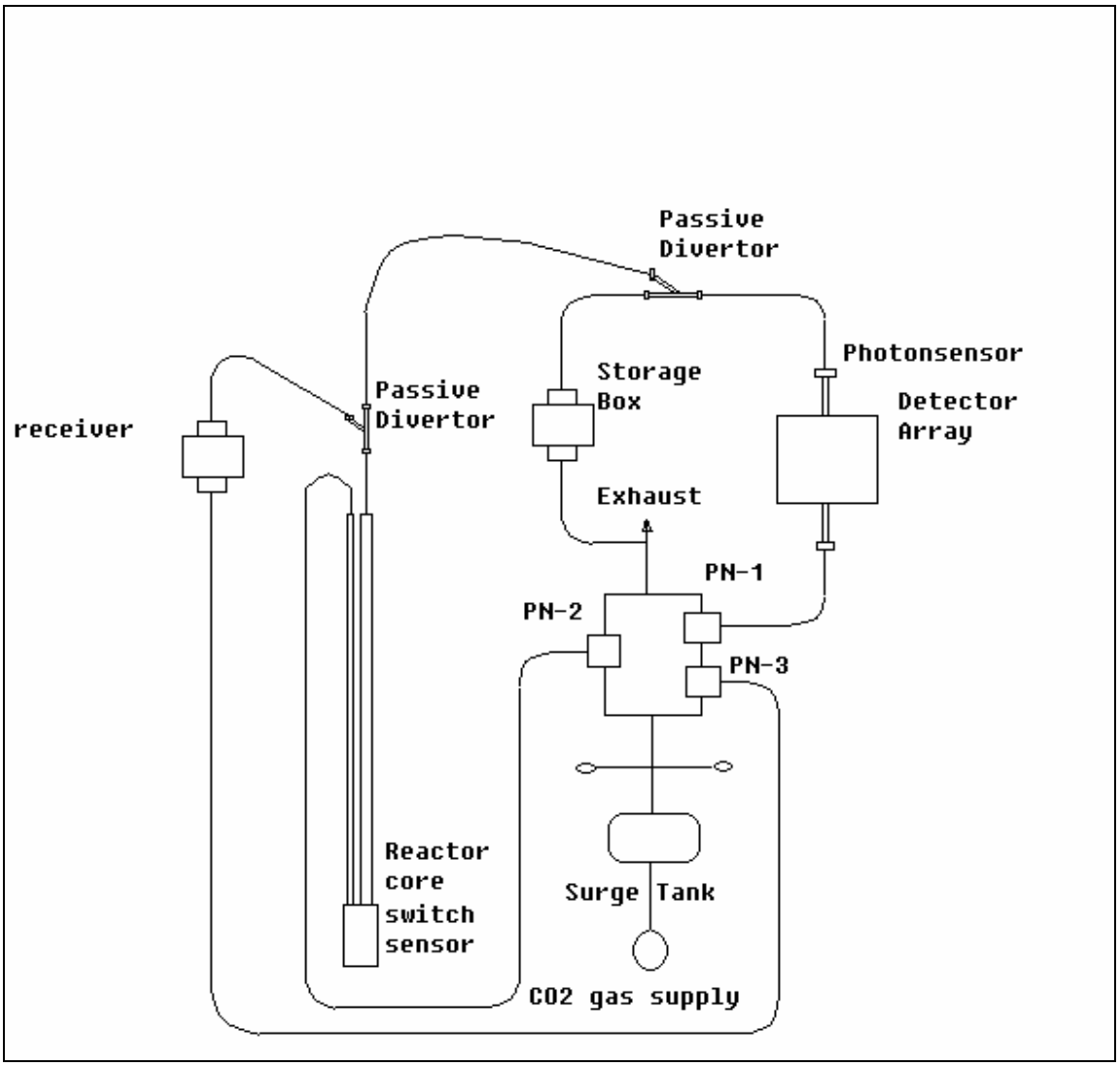


Fig. 6. Fast pneumatic transfer system schematic (PN-1, PN-2 and PN-3 are electrically operated pneumatic valves).

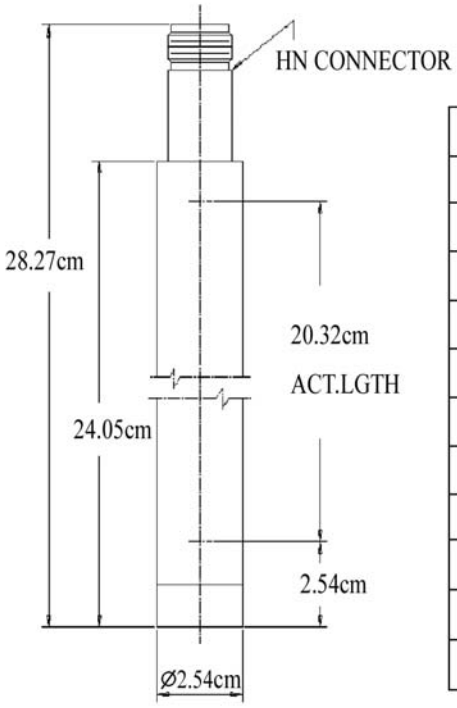
2.2 THE ^{3}He DETECTOR COUNTING STATION

Three ^{3}He tubes (LND Model 252) were embedded in the holes of a graphite-moderated counting station at different positions (Fig. 7 and Fig. 8). To decrease the effects resulted from the gamma rays, 4 cm of lead shot surrounds each detector tube. A thin aluminum can was used to contain the lead shot and fasten the individual detector

in the graphite blocks. Also, because the counting station is constructed with graphite blocks, we can move the blocks to accommodate different detector distances.

The ^{3}He tubes are located at different distances from the sample receiver. By doing this we are able to reconstruct the entire delayed neutron count rate as a function of time for a single radiation. The ^{3}He detectors farthest from the sample will collect the first part of the spectrum when the count rates are high and the nearest detector is paralyzed by the count rate. The ^{3}He detectors closet to the sample will collect the spectrum near the end of the counts when the count rate is low and the far detectors produce few counts. The relative efficiency of the detectors was measured by swapping the ^{3}He detectors among the positions while exposing them to an AmBe neutron source.

The counting station was constructed using graphite to minimize any effects of the delayed neutron energy spectrum. We used the GEANT4 code to simulate the detector efficiencies for different distances and different neutron energies. The code is listed in Appendix A.



Gas Pressure (Torr)	3040
Cathode Material	Aluminum
Effective Diameter (cm)	2.36
Operating Temperature Range (°C)	-50 to +100
Effective Volume (cm^3)	89.01
Sensitivity (cps/nv)	21.8
Recommended Operating Voltage (volts)	1150
Operating Voltage Range (volts)	1050-1400
Maximum Plateau Slope (% / 100 volts)	1
Maximum Resolution (% FWHM)	6
Tube Capacitance (pf)	8
Weight (grams)	142

Fig. 7. The ^{3}He detector (LND model 252).

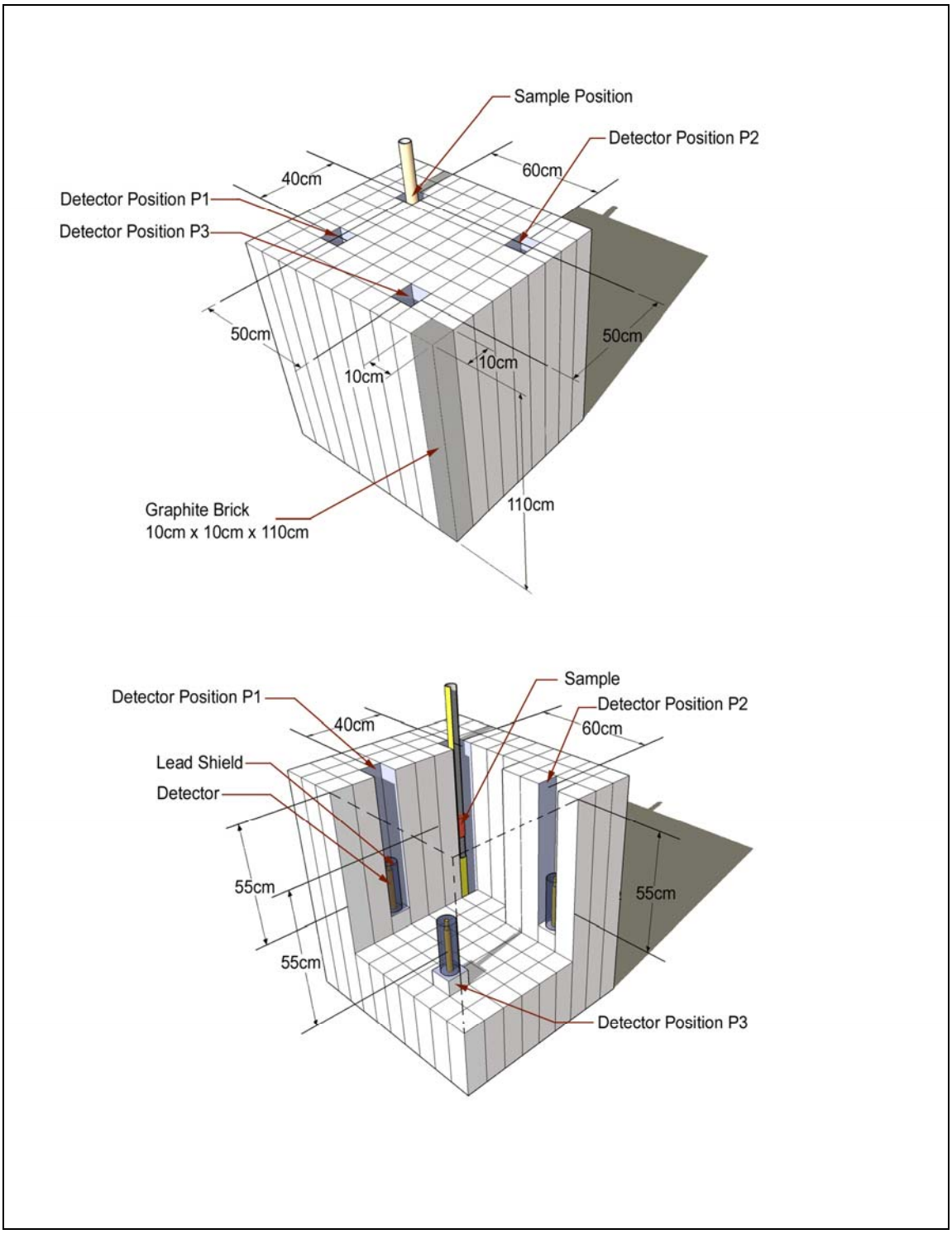


Fig. 8. The ^{3}He detector array (with exploded view).

Fig. 9 shows the detector efficiency (the ratio of the number of the recoil protons divided by the number of the initial neutrons) as a function of monoenergetic neutron energy for selected energies. Note that the detection efficiency for three positions is constant in the delayed neutron energy range from 100 keV to 1000 keV.

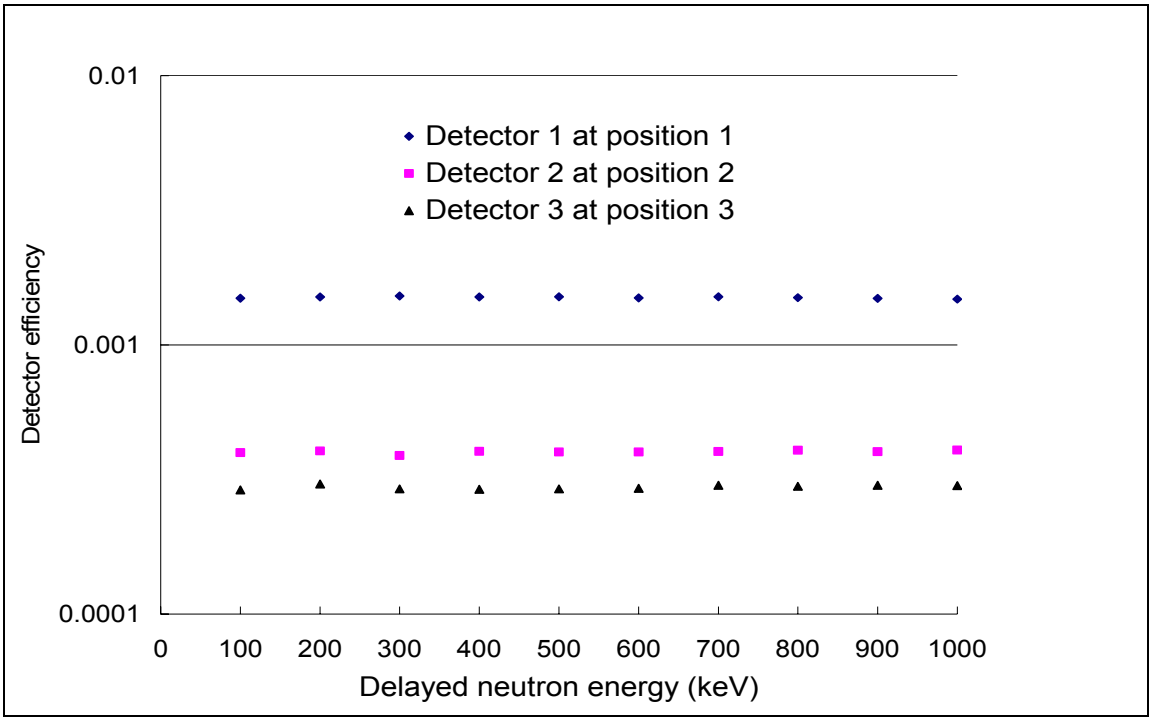


Fig. 9. Detector efficiency for three detectors (position 1 is 40 cm from the receiver; position 2 is 60 cm from the receiver and position 3 is 70 cm from the receiver).

The relative efficiencies of the detectors in different positions were measured by putting individual detector in same position against a constant AmBe neutron source. The calibration time is 30 min for each detector. The results are shown in Table 4.

Table 4. Relative efficiency of the detectors (position 1 is 40 cm from the receiver; position 2 is 60 cm from the receiver and position 3 is 70 cm from the receiver).

	Position 1	Position 2	Position 3
Detector 1	1.01±0.002	1.0±0.003	1.01±0.003
Detector 2	1.0±0.001	1.0±0.002	1.0±0.002
Detector 3	1.02±0.002	1.03±0.003	1.03±0.002

2.3 ELECTRONICS SETUP

The system electronics are shown in Fig.10. Three ^{3}He detectors were embedded in the downstairs graphite-moderated counting station. Since the graphite is electric conductive material, a special designed stander for three preamplifiers (CANBERRA model 2006), high voltage supply connectors and sensor connectors was placed near the counting system and the reactor pool at the upstairs level to keep certain distance from the detectors and the preamplifiers connected with high voltage supply cables. The amplifiers, high voltage supplies and the Multiport II MCA from CANBERRA with three ports were placed in a counting room near the reactor bridge at upstairs level.

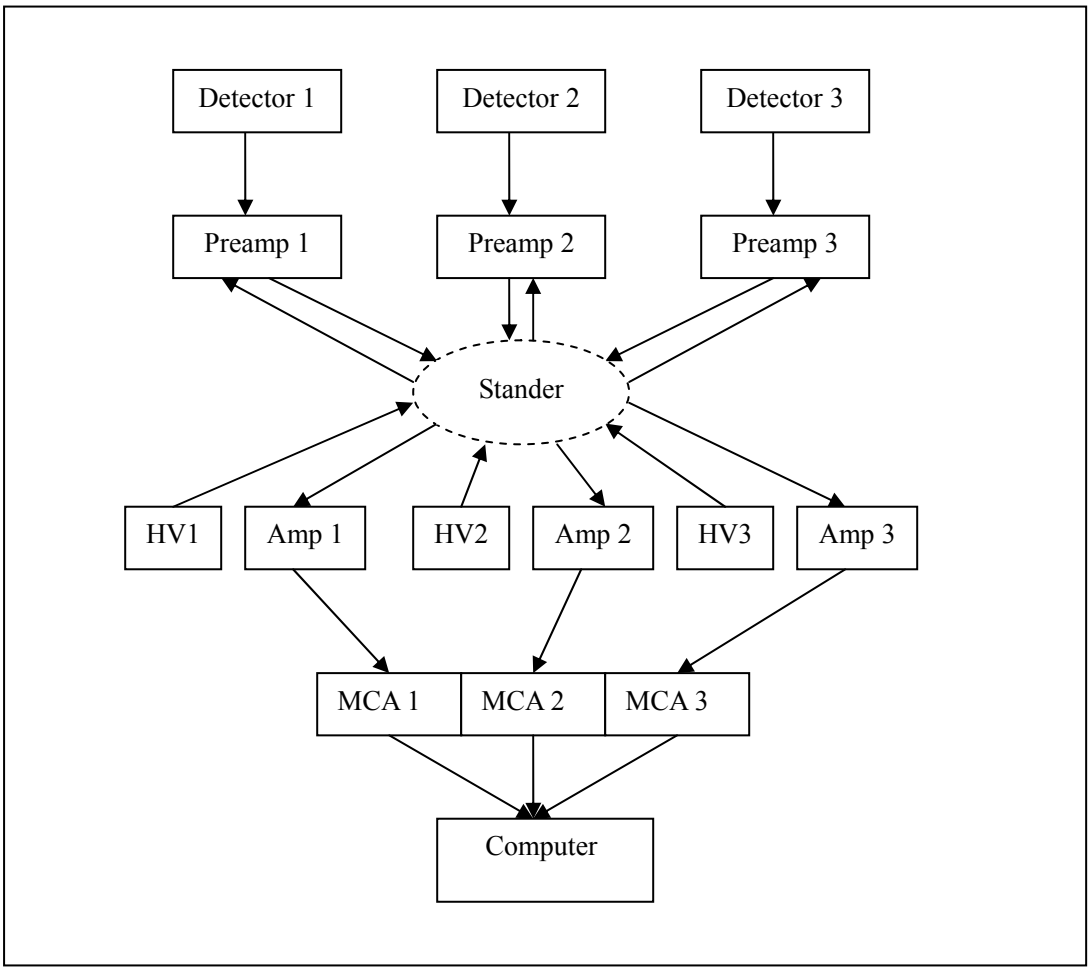


Fig. 10. System electronics.

A series of cables (almost 100 feet for each) runs from the counting station to the counting room around the reactor wall. The Multiport II MCA is fully remote-controlled under Genie 2000 via standard Genie 2000 tools. The distance from high voltage supplies to the stander of preamplifiers was around 100 feet. To maintain the 1000 volt working voltage required by the detector, a large (2 cm) diameter cable was substituted for the normal (0.6 cm) diameter coaxial signal cable.

2.4 INTEGRATED COMPUTER CONTROL SYSTEM

A C language program was used to control the operations of the three solenoids to effectively transfer the sample to and from the reactor and collect the signals from the sensors installed on the sample's track to measure the transit time.

As shown in Fig.11, the C language program was written (with the cooperation of Alfred Hanna) to automatically control the irradiations. The copy of this C program is given in Appendix B. This code allows for three settings: two sensors, one sensor or no sensor. If the program is expected in the "no sensor" and "one sensor" mode, the code assigns a 1 s flight time. The program prompts the user to input the specific irradiation parameters (irradiation time and total counting time), then executes the delayed neutron measurement. First, the sample is transferred into the core from the receiver, where it is irradiated for the specified time period. Then, the sample is transferred to the counting station, and the actual sample transit time is recorded (in the "two sensor" mode). When the sample reaches the counting station (noted by the photosensor), the MCA begins counting with the specified dwell time. After reaching the total counting time, the sample is transferred from the counting station to a remote storage location in the Nuclear Science Center lower research level. The program will read the permit signal from the reactor control room and the initial signal from the sensors. If the readings of the system setting are not in agreement with the user's specification, the program terminates.

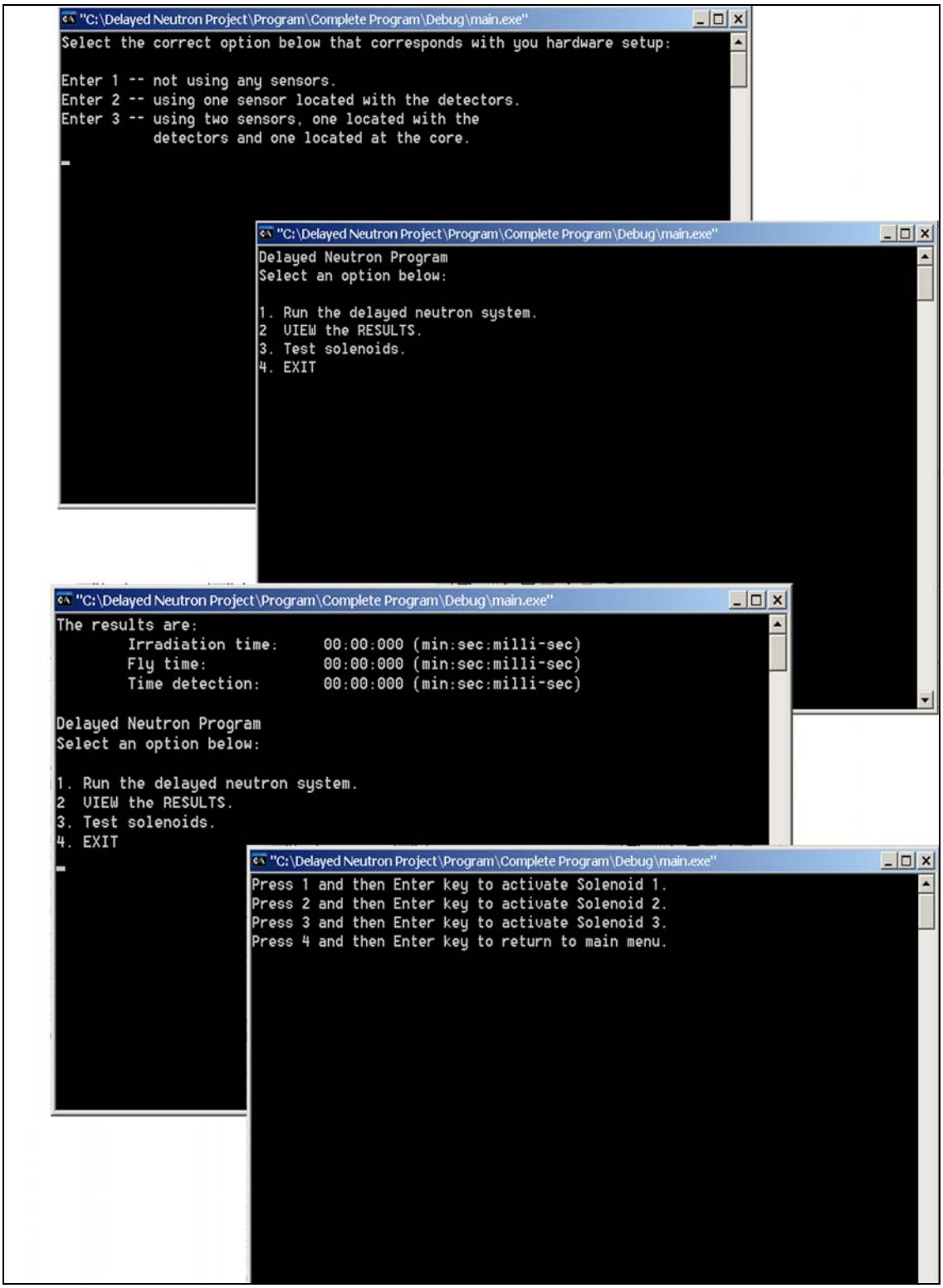


Fig. 11. Computerized control system for pneumatic transfer system.

CHAPTER III

PROCEDURE

The samples were irradiated in the NSCR core at a power lever of 100 kW and an irradiation time of 180 s. All the irradiations were conducted in position B-1 of the NSCR which has a highly thermalized neutron fluence rate. After the irradiation, the samples were transferred to the ^{3}He detectors counting station. The count rates in pre-selected dwell time (25 milliseconds) were collected by the MCAs. The total counting time is 200 s. The flight times were recorded by the control program with a precise timer. When the measurements finished, the samples were then transferred to a remote storage location for further decay. All of these experiments were controlled by a C program.

Three irradiations were performed for each of the U-235 and Pu-239 samples and four blank irradiations to test the system. These samples were fabricated by Oakridge National Laboratory in Tennessee and used by Charlton (1998) in his experiments. The samples were pressed oxide aluminum pellet and were contained in a weld titanium capsule as shown in Fig. 12. We welded the samples in a small plastic tube with foam stuffing for protecting them during the transfer process and avoiding any contamination to the pneumatic system. Table 5 shows the pertinent information of each sample.

Table 5. Samples used in this work.

Element	Sample Number	Actinide Mass (mg)	Purity (%)	Assay Date
U-235	397-22-5	12.27	97.663 ± 0.003	Oct 1,1987
U-235	397-22-6	11.95	97.663 ± 0.003	Oct 1, 1987
Pu-239	114-57-3	10.0	99.745 ± 0.003	Sept 1, 1978
Al blank		48.2	100.0	n/a

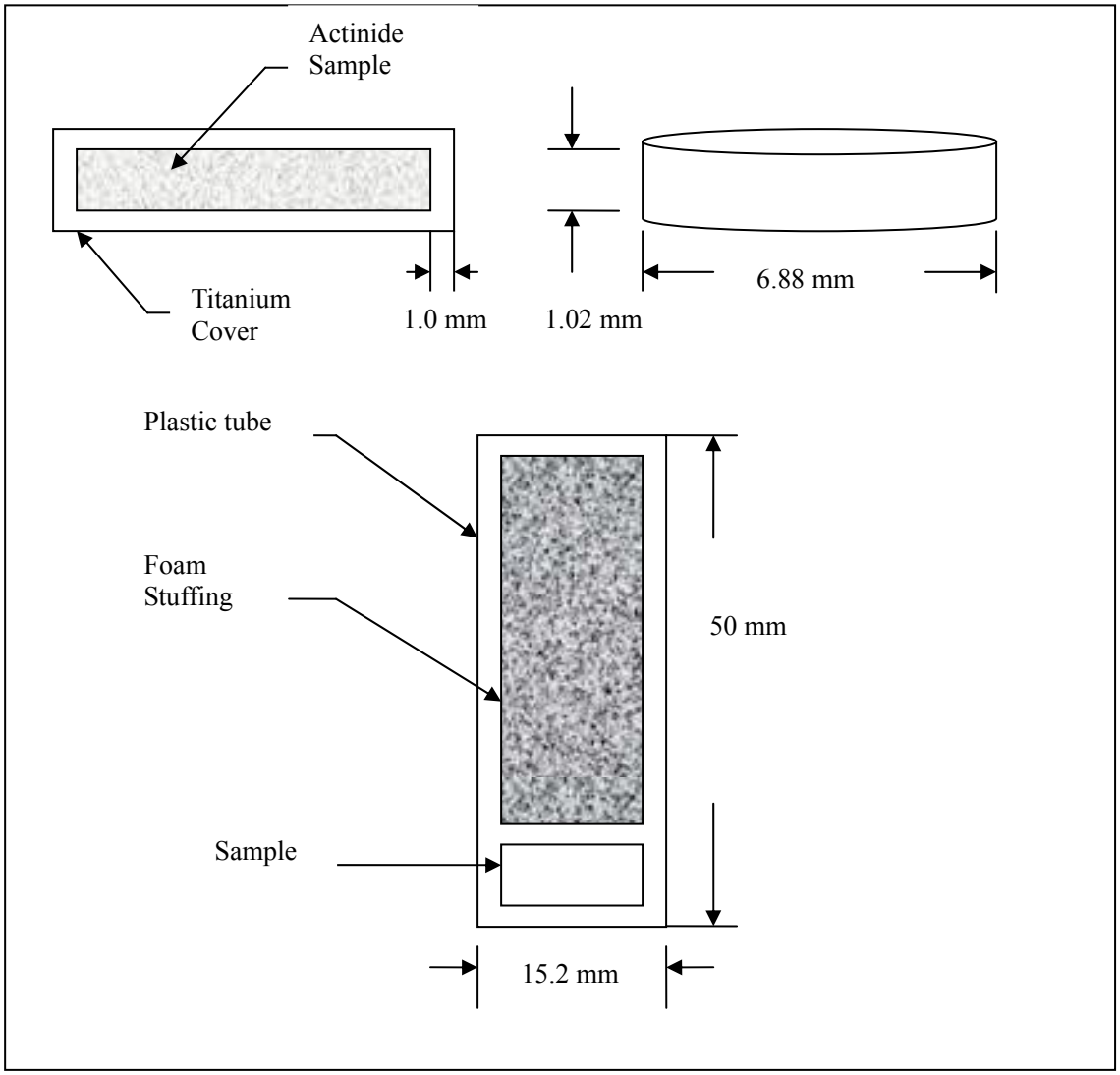


Fig. 12. Sample design.

Fig. 13 shows the graphite counting station and Fig. 14 shows the electronics and computer control of the counting station.

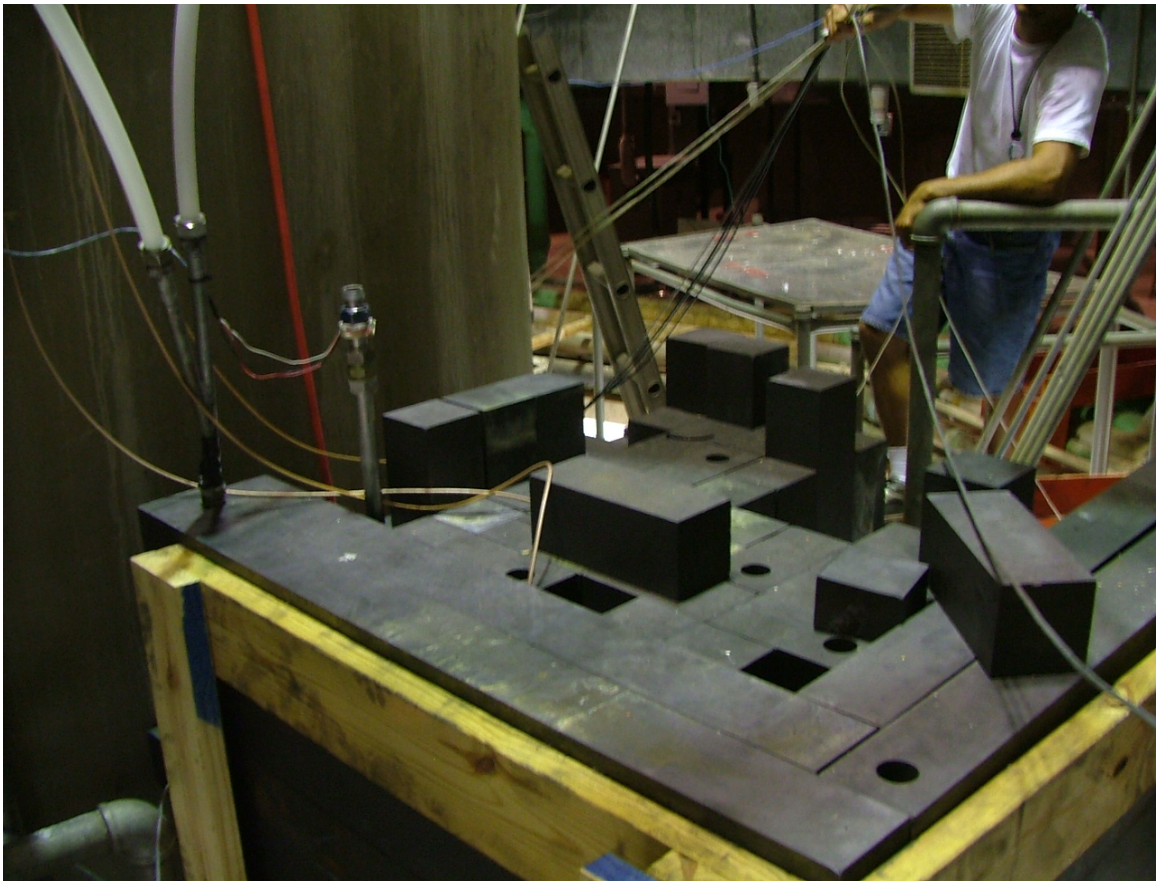


Fig. 13. The counting station.

Fig. 14. The electronics and computer control of the counting system.

CHAPTER IV

RESULTS AND CONCLUSIONS

The measured delayed neutron emission rates equal to the measured time-dependent count rates for each detector divided by the relative detector efficiencies and individual detector energy efficiencies. The emission rates measured by the three detectors were merged together. Detector 1 with the shortest distance to the sample had the highest measured count rates. However it had very large dead time during the early part of the measurement. On the other hand, detector 3 with the longest distance to the sample had lower measured count rates. But it had essentially no dead time. During the decay of the samples, the dead time of each detector decreased with the decrease of gamma rays and delayed neutron emission rates. We separated the total detection time into three parts (0 s to 10 s, 10 s to 50 s, and 50 s to 200 s). For the first part, we used the count rates measured by detector 3, detector 2 for the second part, and detector 3 for the third part.

Parameters (relative yields and decay constants) for five of the traditional six-groups were acquired from the count rates by using a least squares fitting technique developed by Reece and Wang (2005). Eq. (9) was used as the model for the parameter fitting.

Fig. 15 shows the measured delayed neutron count rates for U-235. The sample flight time was 1.124 s.

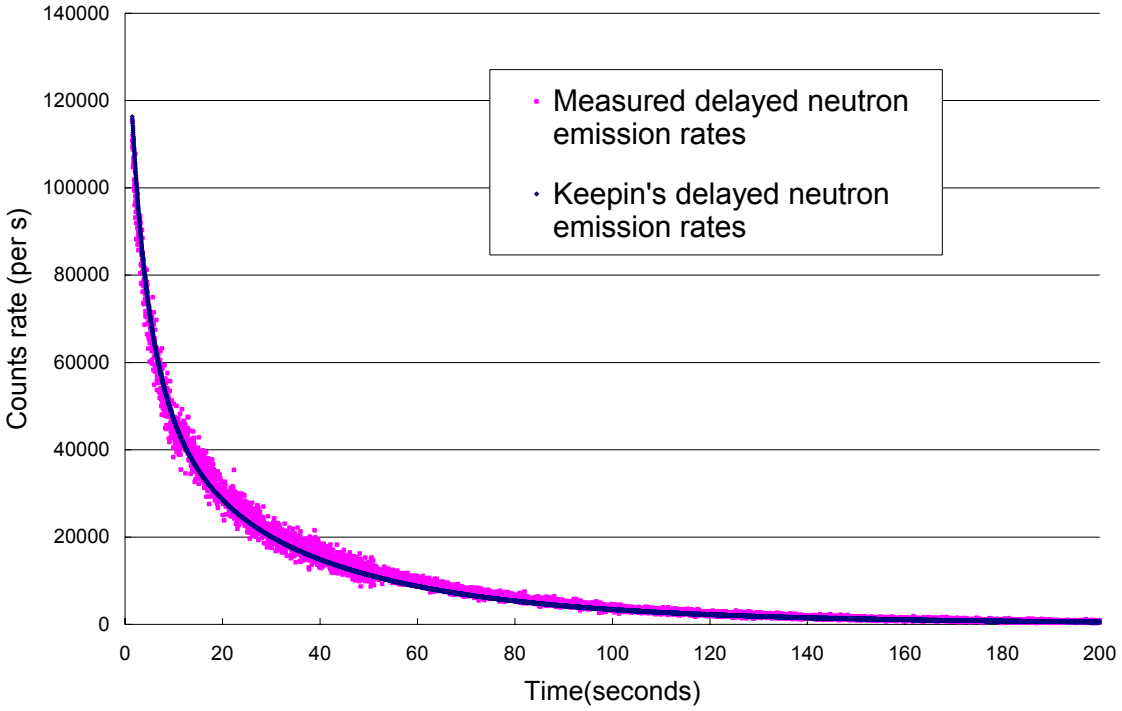


Fig. 15. Measured delayed neutron emission rates (s^{-1}) for U-235.

To compare our measured delayed neutron emission rates and literature values, we separated the time scale in Fig. 15 into three parts: 0-10 seconds, 10-50 seconds, and 50-200 seconds.

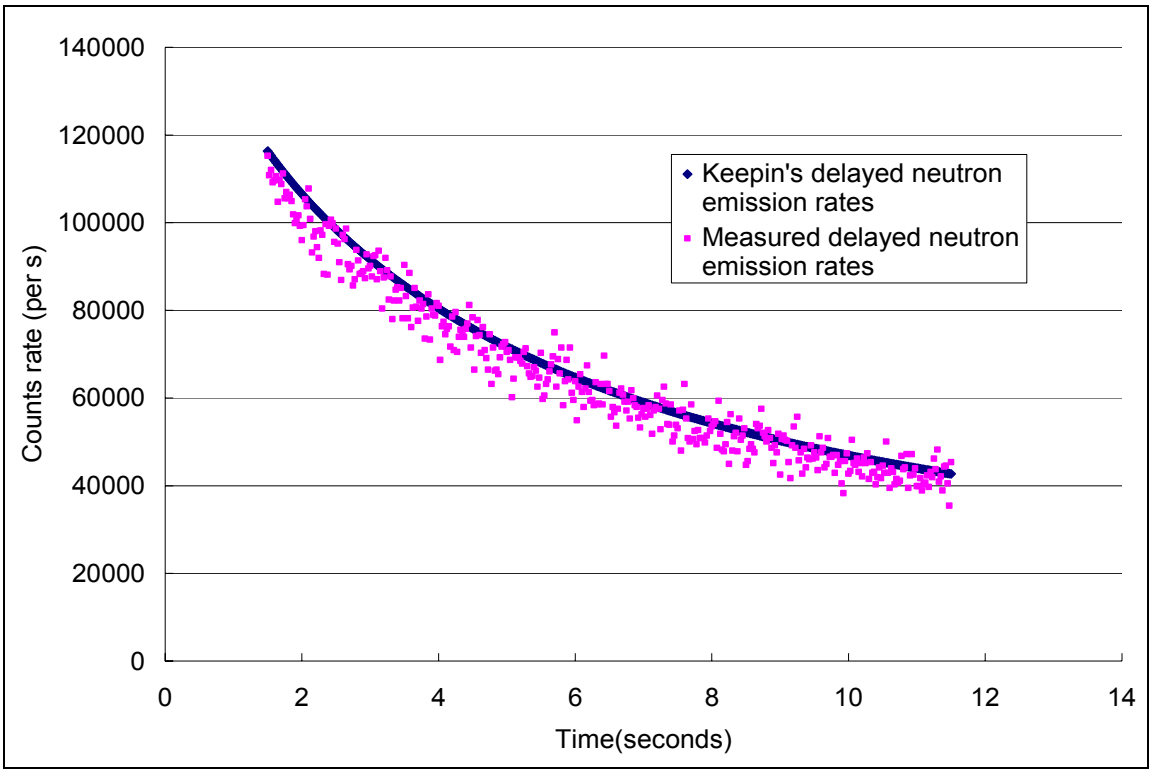


Fig. 16. Measured delayed neutron emission rates (s^{-1}) for U-235 (0 s-10 s).

Fig. 16 shows the measured delayed neutron emission rate at the beginning of the counting period. We know that delayed neutron group 5 and group 6 with short half-lives dominate at this time. Our measurement values appear to lower than Keepin's values.

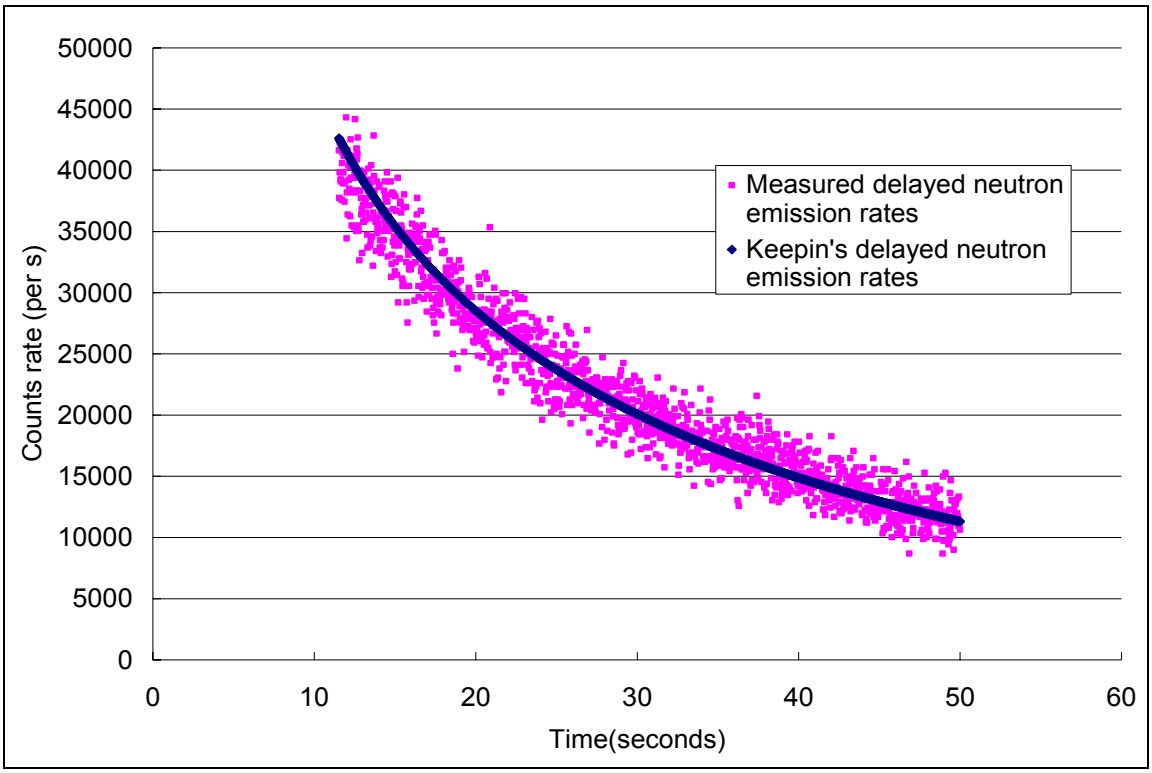


Fig. 17. Measured delayed neutron emission rates (s^{-1}) for U-235 (10 s-50 s).

Fig.17 shows the measured delayed neutron emission rate in the middle of the counting period. We know that delayed neutron group 2, 3 and 4 dominate at this time. Our measured values seem to match Keepin's values.

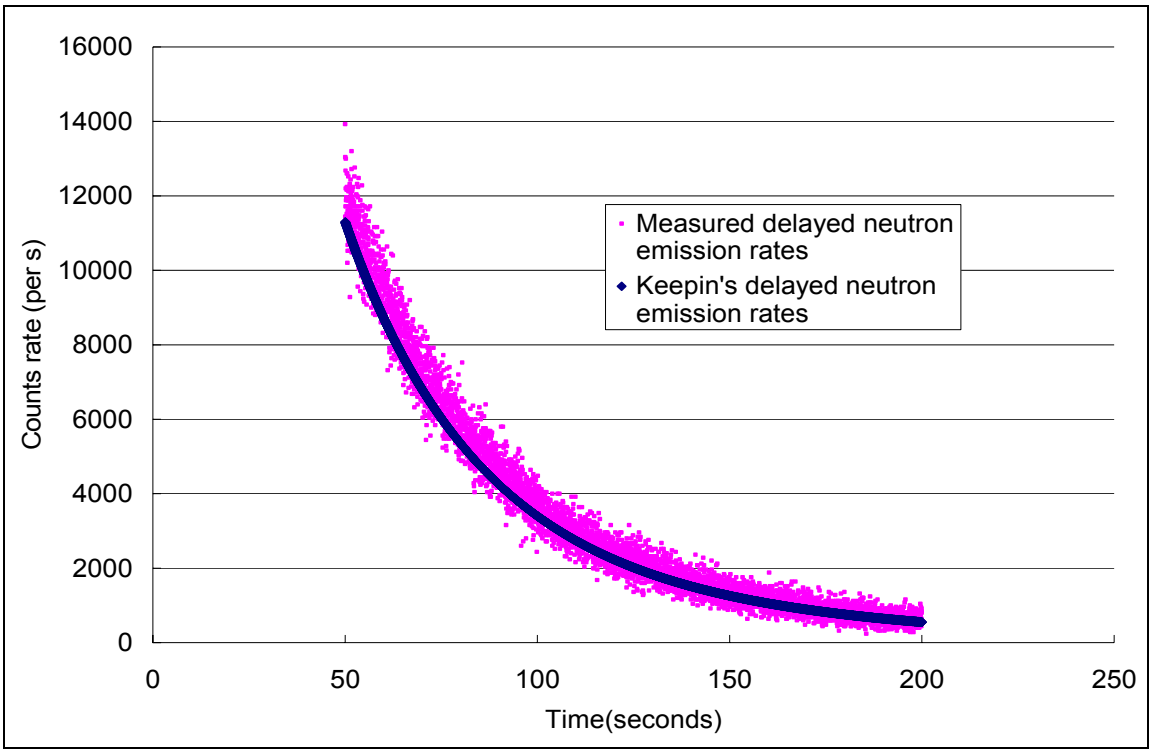


Fig. 18. Measured delayed neutron emission rates (s^{-1}) for U-235 (50 s-200 s).

Fig. 18 shows the measured delayed neutron emission rate at the end of the counting period. We know that delayed neutron group 1 dominates at this time. Our measurements appear to be higher than those calculated values using Keepin's values.

These differences may come from the energy-dependent detector efficiencies for the different counting stations used. Keepin et al. (1957) used paraffin as moderating material. The neutron absorption cross section for the hydrogen is much larger than that for carbon. Because of the difference in thermal neutron absorption, Keepin's counting system may have a higher detector efficiency for the high-energy delayed neutron group 6 and lower detector efficiency for the low-energy delayed neutron group 1 compared with our counting system, using graphite.

The measured delayed neutron emission rate for Pu-239 is shown in Fig.19. The sample flight time is 1.203 s.

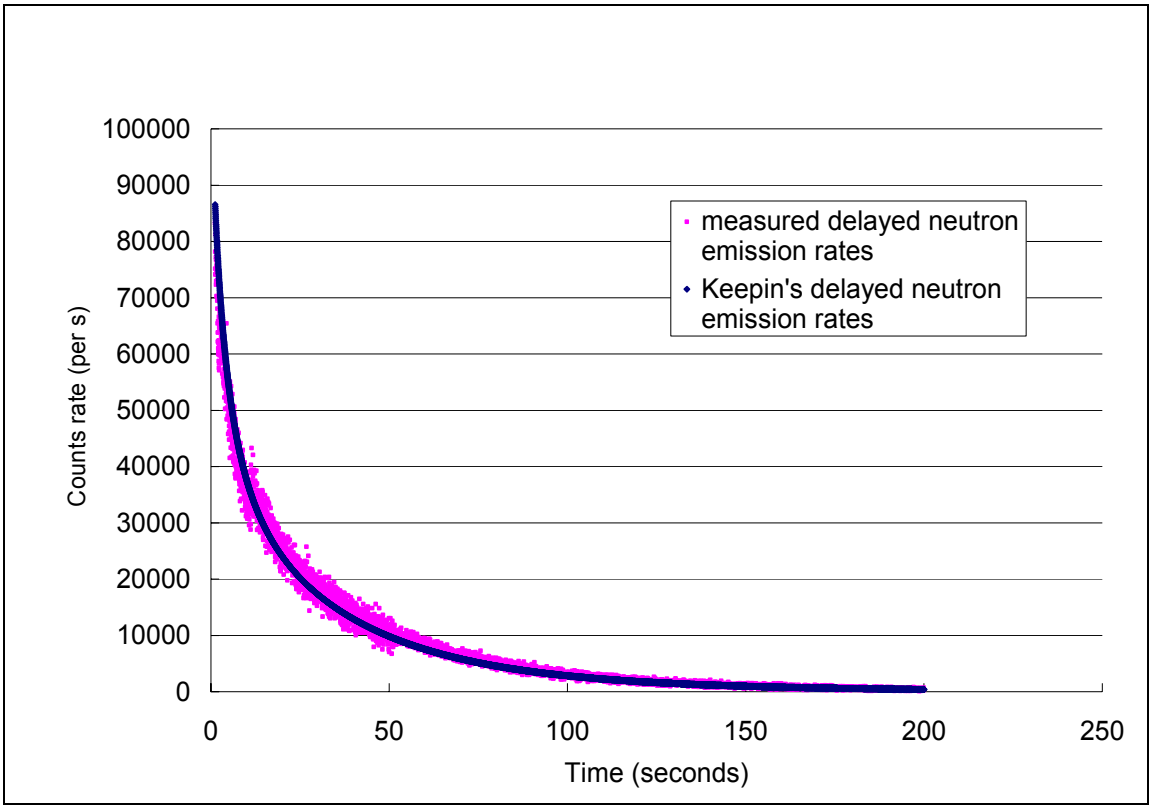


Fig. 19. Measured delayed neutron emission rates (s^{-1}) for Pu-239.

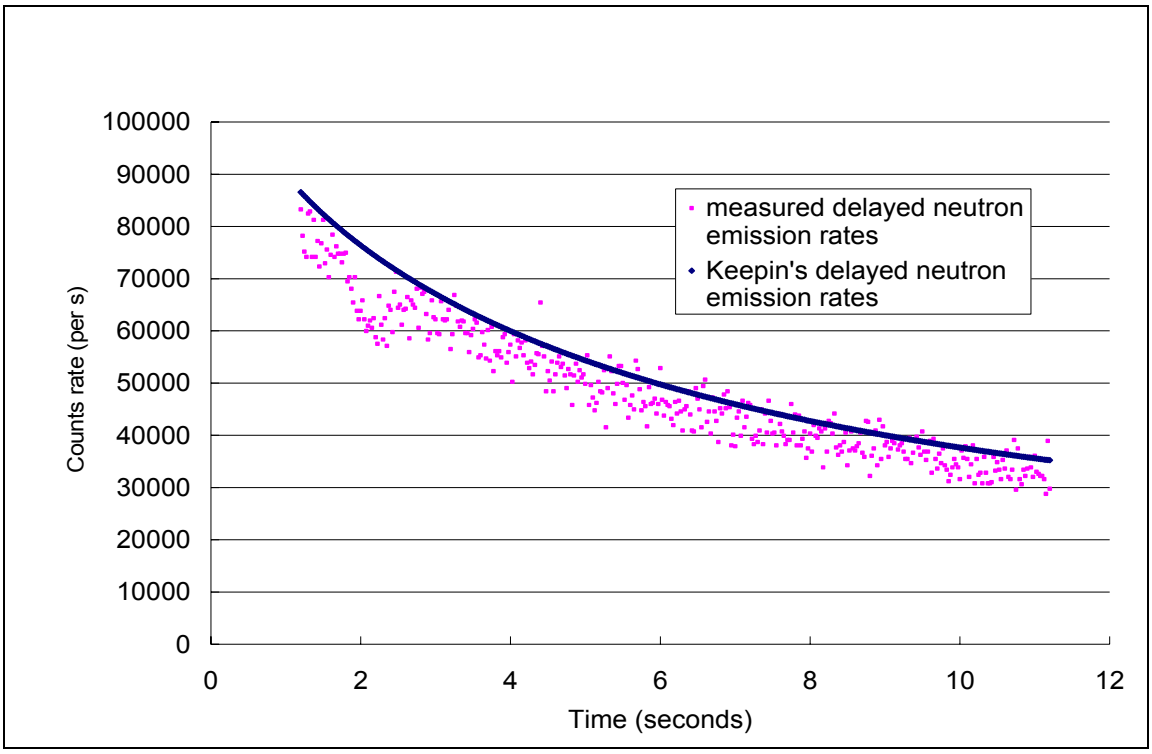


Fig. 20. Measured delayed neutron emission rates (s^{-1}) for Pu-239 (0 s-10 s).

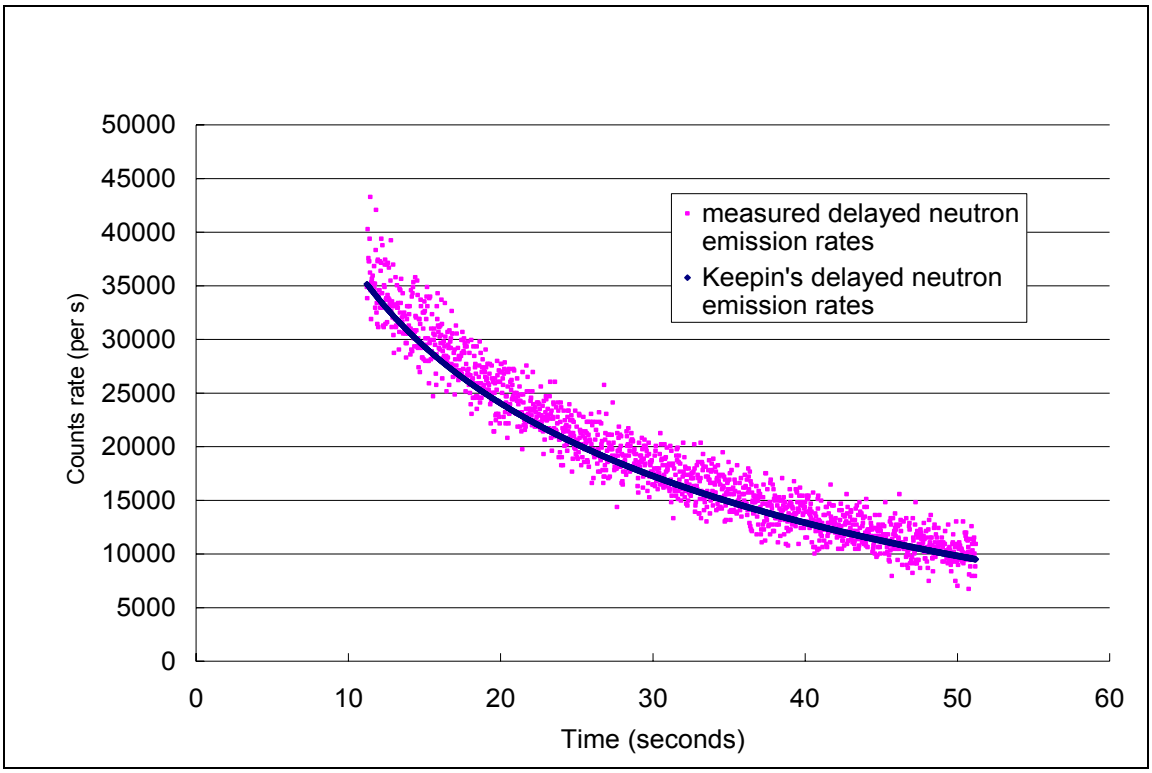


Fig. 21. Measured delayed neutron emission rates (s^{-1}) for Pu-239 (10 s-50 s).

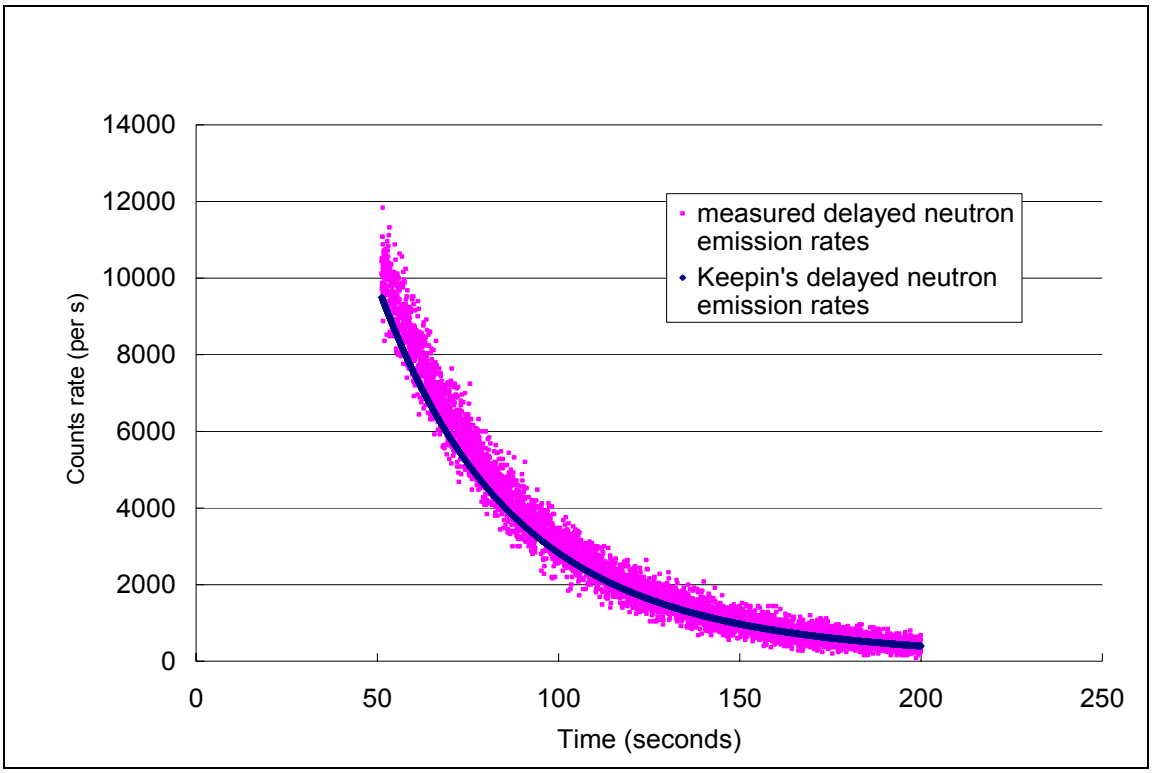


Fig. 22. Measured delayed neutron emission rates (s^{-1}) for Pu-239 (50 s-200 s).

We see roughly the same pattern of response for Pu-239 as we did for U-235 which were shown in Fig.20, Fig.21 and Fig.22. The relative yields were then normalized to unity in order to match the formulation of Eq. (9). The equation for normalization is as follows,

$$\alpha_j = \frac{A_j}{\sum_{j=1}^{6} A_j}, \tag{18}$$

where α_j is the normalized relative yield and A_j is determined above. Table 6 contains the values of the group decay constants and the relative group yields in the five-groups structure for U-235 and Pu-239.

Table 6. Measured delayed neutron parameters (decay constants and relative yields) for U-235 and Pu-239.

	U-235			Pu-239		
	This work	Keepin et al. 1957	Waldo et al. 1981	This work	Keepin et al. 1957	Waldo et al. 1981
λ_1	0.0121	0.01244	0.01255	0.0122	0.0128	0.01246
α_1	0.033	0.033	0.033	0.027	0.035	0.0269
λ_2	0.0317	0.0305	0.0309	0.0311	0.03	0.02941
α_2	0.223	0.219	0.205	0.278	0.298	0.259
λ_3	0.08	0.1114	0.114	0.092	0.12375	0.0714
α_3	0.145	0.196	0.199	0.082	0.211	0.111
λ_4	0.275	0.3013	0.328	0.22	0.31	0.212
α_4	0.41	0.395	0.388	0.204	0.326	0.2246
λ_5	2.09	1.136	2.06	0.345	1.1214	0.324
α_5	0.158	0.115	0.175	0.229	0.086	0.209
λ_6		3.013		1.26	2.6965	1.28
α_6		0.042		0.149	0.044	0.169

The measurements by Keepin et al. (1957) and Waldo et al. (1981) were performed in a thermal irradiation position. The values measured here are close to those of Waldo et al. (1981). However, we do see minor differences between our data and the data of Keepin et al. (1957), particularly in the first 10 seconds after irradiation and after 150 seconds post-irradiation.

CHAPTER V

SUMMARY AND FUTURE WORK

A new delayed neutron counting system has been installed at the Nuclear Science Center using graphite as the moderating medium in the counting station. The pneumatic transfer and the operation of the counting system is computer controlled. The flight times can be measured to within a millisecond. Knowing the flight times accurately allows the combination of multiple runs with minimal error. By combining multiple runs the effects of Poisson statistical variation can be minimized. Furthermore, because of the varying distances of the detectors from the counting station receiver, the count rate can be collected from entry into the counting station out to 200 seconds and beyond for each sample. Detector 1 with the shortest distance to the sample is saturated at the beginning of the counting period because of high counts rate. However, Detector 3 with the longest distance to the sample works well at this time. At the end of the counting period, detector 3 collects few counts because of the distance from the samples, but detector 1 works well because it is closer. The relative efficiencies for individual detectors were measured carefully so the output of the three detectors could be combined.

Initial runs have been made with U-235 and Pu-239 and our data suggest that the system is working properly.

Several tasks remain that can improve the system. The shortest flight time achieved so far is around 600 milliseconds. Because group 6 has an average half life of around 300

milliseconds, the signal from this group has already died to about 25% of its original value. Efforts to decrease the flight time should continue. Another important advance would be to pulse the reactor and then transfer the sample to the counting station. This would minimize the effects of the long-lived precursors and a much stronger group 6 signal could be obtained.

With this new data, further work could be done on resolution of mathematical methods to describe delayed neutron counts rate as a function time and, hopefully, any bias in the literature values could be corrected.

REFERENCES

Bohr N, Wheeler JA. The mechanism of nuclear fission. Phys Rev 56:426-450; 1939.

Brady MC, England TR. Fission-product chain yields and delayed neutrons. Nuclear Science and Engineering 129:103-109; 1989.

Charlton WS. Delayed neutron emission measurements from fast fission of actinide waste isotopes. College Station, TX: Department of Nuclear Engineering, Texas A&M University; 1998. Thesis.

European Organization for Nuclear Research. GEANT4 website [online]. Available at: http://geant4.web.cern.ch/geant4/support/index.shtml. Accessed 12 June 2005.

Gill PE, Murray W. Quasi-Newton methods for unconstrained optimization. J Appl Math 9(1):91-108; 1972.

Hanson AO, McKibben J. A neutron detector having uniform sensitivity from 10 keV to 3 MeV. Phys Rev 72:673–677; 1947.

Jewell RW, John W, White DH. A high efficient graphite-moderated neutron counter. Nuclear Instruments and Methods 63:185-188; 1968.

Keepin GR, Wimett TF, Zeigler RK. Delayed neutrons from fissionable isotopes of uranium, plutonium, and thorium. Phys Rev 107:1044-1049; 1957.

Levenberg K. A method for the solution of certain problems in least squares. J Appl Math 2:164-168; 1944.

Marquardt D. An algorithm for least-squares estimation of nonlinear parameters. J Appl Math 11:431-441; 1963.

Reece WD, Wang J. Assessment of uncertainty in the literature values for the delayed neutron precursor groups. College Station, TX: Department of Nuclear Engineering, Texas A&M University; 2005. Report for Sandia National Laboratories.

Roberts R, Meyer R, Hafstad L, Wang P. The delayed neutron emission which accompanies fission of uranium and thorium. Phys Rev 55:664; 1939.

Saleh H. Neutron emission measurements from actinide waste isotopes. College Station, TX: Department of Nuclear Engineering, Texas A&M University; 1995. Dissertation.

Visual Numerics. IMSL FORTRAN Numerical Library website [online]. Available at: http://www.vni.com/products/imsl/fortran/overview.html. Accessed 12 June 2005.

Waldo RW, Karam RA, Meyer RA. Delayed neutron yields: time dependent measurements and a predictive model. Phys Rev 23:1113-1127; 1981.

APPENDIX A

GEANT4 CODE FOR DETECTOR ARRAY

```
// --------------------------------------------------------------
//      GEANT 4 - A01app program written by Yong Chen
//      main.c
// --------------------------------------------------------------
#include "G4RunManager.hh"
#include "G4UImanager.hh"
#include "G4UIterminal.hh"
#include "G4UItcsh.hh"
#include "A01DetectorConstruction.hh"
#include "LHEP_PRECO_HP.hh"
#include "A01PrimaryGeneratorAction.hh"
#include "A01TrackingAction.hh"
#include "A01SteppingAction.hh"
#include "A01EventAction.hh"
#include "A01RunAction.hh"
#ifdef G4VIS_USE
#include "A01VisManager.hh"
#endif

int main(int argc,char** argv)
{
      // RunManager construction
  G4RunManager* runManager = new G4RunManager;

#ifdef G4VIS_USE
  // Visualization manager construction
  G4VisManager* visManager = new A01VisManager();
  visManager->Initialize();
#endif

  // mandatory user initialization classes
  runManager->SetUserInitialization(new A01DetectorConstruction);

  //LHEP * thePList = new LHEP;
  LHEP_PRECO_HP * thePList = new LHEP_PRECO_HP;

  runManager->SetUserInitialization(thePList);

  // initialize Geant4 kernel
  runManager->Initialize();

  // mandatory user action class
  runManager->SetUserAction(new A01PrimaryGeneratorAction);

  // optional user action classes
  runManager->SetUserAction(new A01RunAction);
  runManager->SetUserAction(new A01EventAction);
  runManager->SetUserAction(new A01SteppingAction);
  runManager->SetUserAction(new A01TrackingAction);

  if(argc>1)
  // execute an argument macro file if exist
  {
    G4UImanager* UImanager = G4UImanager::GetUIpointer();
    G4String command = "/control/execute ";
    G4String fileName = argv[1];
    UImanager->ApplyCommand(command+fileName);
  }
  else
```

```
  // start interactive session
  {
    G4UIsession* session = new G4UIterminal();
    session->SessionStart();
    delete session;
  }

#ifdef G4VIS_USE
  delete visManager;
#endif

  delete runManager;

  return 0;
}
// --------------------------------------------------------------
//      GEANT 4 - A01app
//      A01DetectorConstruction.c
// --------------------------------------------------------------

#include "A01DetectorConstruction.hh"
#include "G4Material.hh"
#include "G4Element.hh"
#include "G4MaterialTable.hh"
#include "G4Isotope.hh"
#include "G4VSolid.hh"
#include "G4Box.hh"
#include "G4Tubs.hh"
#include "G4LogicalVolume.hh"
#include "G4VPhysicalVolume.hh"
#include "G4PVPlacement.hh"
#include "G4PVParameterised.hh"
#include "G4SDManager.hh"
#include "G4VSensitiveDetector.hh"
#include "G4RunManager.hh"
#include "G4VisAttributes.hh"
#include "G4Colour.hh"
#include "G4ios.hh"

A01DetectorConstruction::A01DetectorConstruction()
{;}
A01DetectorConstruction::~A01DetectorConstruction()
{;}

G4VPhysicalVolume* A01DetectorConstruction::Construct()
{
// constructMaterials

  G4double a;
  G4double z;
  G4double density, temperature, pressure;
  G4double weightRatio,abundance;
  G4String name;
  G4String symbol;
  G4int nElem,natoms,iz,n,ncomponents;

  // elements for mixtures and compounds

  a = 14.01*g/mole;
```

```
  G4Element* elN = new G4Element(name="Nitrogen", symbol="N", z=7., a);
  a = 16.00*g/mole;
  G4Element* elO = new G4Element(name="Oxigen", symbol="O", z=8., a);
  a = 10.81*g/mole;
  G4Element* elB = new G4Element(name="Boron", symbol="B", z=5., a);
  a = 18.99*g/mole;
  G4Element* elF= new G4Element(name="Flourine", symbol="F", z=9., a);
  a = 2*g/mole;
  G4Element* elH= new G4Element(name="Hy", symbol="H", z=1., a);

  G4Isotope* He3 = new G4Isotope(name="He3", iz=2, n=3, a=3*g/mole);
  G4Element* elHe  = new G4Element(name="helium3", symbol="He", ncomponents=1);
  elHe->AddIsotope(He3, abundance= 100.*perCent);

  // Air
  density = 1.29*mg/cm3;
  air = new G4Material(name="Air", density, nElem=2);
  air->AddElement(elN, weightRatio=0.7);
  air->AddElement(elO, weightRatio=0.3);
 //He3 gas

  density = 0.7144*mg/cm3;
  pressure = 405.3e+3*pascal;
  temperature = 300.*kelvin;
  He3Gas = new G4Material(name="He3Gas", density, nElem=1, kStateGas, temperature, pressure);
  He3Gas->AddElement(elHe, natoms=1);

 // graphite
  a = 12.01*g/mole;
  density = 2.1*g/cm3;
  graphite = new G4Material(name="graphite", z=6., a, density);

  //vaccum
  density     = universe_mean_density;               //from PhysicalConstants.h
  pressure    = 1.e-19*pascal;
  temperature = 0.1*kelvin;
  G4Material* Galactic = new G4Material(name="Galactic", z=1., a=1.01*g/mole, density,
                kStateGas,temperature,pressure);

  G4cout << G4endl << "The materials defined are : " << G4endl << G4endl;
  G4cout << *(G4Material::GetMaterialTable()) << G4endl;


  // geometries ---------------------------------------------------------------
  // experimental hall (world volume)
  G4VSolid* worldSolid = new G4Box("worldBox",2.0*m,2.0*m,2.0*m);
  G4LogicalVolume* worldLogical
    = new G4LogicalVolume(worldSolid,Galactic,"worldLogical",0,0,0);
  G4VPhysicalVolume* worldPhysical
    = new G4PVPlacement(0,G4ThreeVector(0,0,0),worldLogical,"worldPhysical",0,0,0);

  //main box
  G4VSolid* mainSolid = new G4Box("main",100.0/2*cm,100.0/2*cm,110.0/2*cm);
  G4LogicalVolume* mainLogical
    = new G4LogicalVolume(mainSolid,graphite,"mainLogical",0,0,0);
  G4VPhysicalVolume* mainPhysical
    =new G4PVPlacement(0,G4ThreeVector(0,0,0),mainLogical,"mainPhysical",worldLogical,0,0);

  // source position (-25,25,0)
```

```
  //airbox P1
   G4VSolid* airSolid = new G4Box("airbox",10.0/2*cm,10.0/2*cm,110.0/2*cm);
  G4LogicalVolume* airLogical
    = new G4LogicalVolume(airSolid,air,"airLogical",0,0,0);
  G4VPhysicalVolume* airPhysical
    =new
G4PVPlacement(0,G4ThreeVector(35.0*cm,25.0*cm,0.*cm),airLogical,"airPhysical",mainLogical,0,0);

    //He3 tubs
  G4VSolid* He31Solid = new G4Tubs("He31Tubs",0.*m,2.54/2*cm,20.0/2*cm,0.,2.*M_PI);
  G4LogicalVolume* He31Logical
    = new G4LogicalVolume(He31Solid,He3Gas,"He31Logical",0,0,0);
  G4VPhysicalVolume* He31Physical
    =new
G4PVPlacement(0,G4ThreeVector(0.*cm,0.*cm,0.*cm),He31Logical,"He31Physical",airLogical,0,0);

  //P2
   G4VSolid* air2Solid = new G4Box("air2box",10.0/2*cm,10.0/2*cm,110.0/2*cm);
  G4LogicalVolume* air2Logical
    = new G4LogicalVolume(air2Solid,air,"air2Logical",0,0,0);
  G4VPhysicalVolume* air2Physical
    =new
G4PVPlacement(0,G4ThreeVector(-25.0*cm,-15.0*cm,0.*cm),air2Logical,"air2Physical",mainLogical,0,0);

    //He3 tubs
  G4VSolid* He32Solid = new G4Tubs("He32Tubs",0.*m,2.54/2*cm,20.0/2*cm,0.,2.*M_PI);
  G4LogicalVolume* He32Logical
    = new G4LogicalVolume(He32Solid,He3Gas,"He32Logical",0,0,0);
  G4VPhysicalVolume* He32Physical
   =new
G4PVPlacement(0,G4ThreeVector(0.*cm,0.*cm,0.*cm),He32Logical,"He32Physical",air2Logical,0,0);

     //P3
   G4VSolid* air3Solid = new G4Box("air3box",10.0/2*cm,10.0/2*cm,110.0/2*cm);
  G4LogicalVolume* air3Logical
    = new G4LogicalVolume(air3Solid,air,"air3Logical",0,0,0);
  G4VPhysicalVolume* air3Physical
    =new
G4PVPlacement(0,G4ThreeVector(25.0*cm,-25.0*cm,0.*cm),air3Logical,"air3Physical",mainLogical,0,0);

    //He3 tubs
  G4VSolid* He33Solid = new G4Tubs("He33Tubs",0.*m,2.54/2*cm,20.0/2*cm,0.,2.*M_PI);
  G4LogicalVolume* He33Logical
    = new G4LogicalVolume(He33Solid,He3Gas,"He33Logical",0,0,0);
  G4VPhysicalVolume* He33Physical
    =new
G4PVPlacement(0,G4ThreeVector(0.*cm,0.*cm,0.*cm),He33Logical,"He33Physical",air3Logical,0,0);

  // return the world physical volume ----------------------------------------

  G4cout << G4endl << "The geometrical tree defined are : " << G4endl << G4endl;
  return worldPhysical;
}
// ------------------------------------------------------------
//      GEANT 4 - A01app
//      A01DetectorConstruction.c
// ------------------------------------------------------------
#include "A01PrimaryGeneratorAction.hh"
//#include "A01PrimaryGeneratorMessenger.hh"
```

```
#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"
#include "Randomize.hh"
#include "G4Geantino.hh"

A01PrimaryGeneratorAction::A01PrimaryGeneratorAction()
{

  G4int n_particle = 1;
  particleGun  = new G4ParticleGun(n_particle);

  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4String particleName;
  G4ParticleDefinition* particle
                = particleTable->FindParticle(particleName="neutron");
  particleGun->SetParticleDefinition(particle);
  particleGun->SetParticlePosition(G4ThreeVector(-25.0*cm,25.0*cm,0.*cm));
  particleGun->SetParticleEnergy(0.1*MeV);

}

A01PrimaryGeneratorAction::~A01PrimaryGeneratorAction()
{
  delete particleGun;

}

void A01PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
//direction

  G4double xi1 = G4UniformRand();
  G4double xi2;
  G4double xi3;

  //4pi
  G4double cosThita = 2.0*xi1 - 1.0;

  //2pi
  //G4double cosThita = xi1;
  G4double sinThita = sqrt(1.0 - cosThita*cosThita);
  G4double sumofPhi;

  do
  {
       xi2 = 2.0*G4UniformRand()-1.;
       xi3 = G4UniformRand();
       sumofPhi = xi2*xi2+xi3*xi3;
  }while (sumofPhi>1.);

  G4double cosPhi = (xi2 * xi2-xi3 * xi3) / sumofPhi;
  G4double sinPhi = 2.*xi2*xi3/sumofPhi;

  G4double x0 = sinThita*cosPhi;
  G4double y0 = sinThita*sinPhi;
  G4double z0 = cosThita;
```

```
  particleGun->SetParticleMomentumDirection(G4ThreeVector(z0,x0,y0));

  particleGun->GeneratePrimaryVertex(anEvent);

}
// --------------------------------------------------------------
//      GEANT 4 - A01app
//      A01SteppingAction.c
// --------------------------------------------------------------
#include "A01SteppingAction.hh"
//#include "def.h"
#include "G4SteppingManager.hh"
#include "G4Track.hh"
#include "G4Step.hh"
#include "G4StepPoint.hh"
#include "G4TrackStatus.hh"
#include "G4VPhysicalVolume.hh"
#include "G4ParticleDefinition.hh"
#include "G4ParticleTypes.hh"
#include "G4VProcess.hh"

A01SteppingAction::A01SteppingAction()
{;}
A01SteppingAction::~A01SteppingAction()
{;}
int hitsForproton1;
int hitsForproton2;
int hitsForproton3;
void A01SteppingAction::UserSteppingAction(const G4Step * theStep)
{
  G4Track * theTrack = theStep->GetTrack();
  G4int CSD = theTrack->GetCurrentStepNumber();
  G4int PD = theTrack->GetParentID();

  G4ParticleDefinition * particleType = theTrack->GetDefinition();
// proton detection
  if(PD==1)

  {
      if((particleType==G4Proton::ProtonDefinition()))

   {     //G4cout << " get proton" << G4endl;
        G4StepPoint * theNextPoint1 = theStep->GetPreStepPoint();
        G4VPhysicalVolume * theNextPV1 = theNextPoint1->GetPhysicalVolume();
        if(theNextPV1!=NULL)
        {
               G4String theNextPV1name = theNextPV1->GetName();
               if (theNextPV1name(0,4)=="He31")
               {
                  const G4VProcess* process  = theTrack->GetCreatorProcess() ;
                      if (0 != process )
                      { G4String aaa = process->GetProcessName();
                        G4ProcessType bbb = process->GetProcessType();
                            if(CSD==1)
                            {
                           // G4cout << " get proton in He31 " << G4endl;


                                  hitsForproton1 = hitsForproton1 +1;
```

```
                        //G4cout << "Total hitsForproton1: " << hitsForproton1 << G4endl;
                    }

                }

            }

            if (theNextPV1name(0,4)=="He32")
            {
               const G4VProcess* process  = theTrack->GetCreatorProcess() ;
                   if (0 != process )
                   { G4String aaa2 = process->GetProcessName();
                     G4ProcessType bbb2 = process->GetProcessType();
                        if(CSD==1)
                        {
                         //G4cout << " get proton in He32 " << G4endl;


                            hitsForproton2 = hitsForproton2 +1;

                            //G4cout << "Total hitsForproton2: " << hitsForproton2 << G4endl;
                        }

                   }

            }

            if (theNextPV1name(0,4)=="He33")
            {
               const G4VProcess* process  = theTrack->GetCreatorProcess() ;
                   if (0 != process )
                   { G4String aaa3 = process->GetProcessName();
                     G4ProcessType bbb3 = process->GetProcessType();
                        if(CSD==1)
                        {
                         //G4cout << " get proton in He33 " << G4endl;

                            hitsForproton3 = hitsForproton3 +1;

                            //G4cout << "Total hitsForproton3: " << hitsForproton3 << G4endl;
                        }

                   }

             }


        }


   }
  }
// --------------------------------------------------------------
//      GEANT 4 - A01app
//      A01EventAction.c
// --------------------------------------------------------------

#include "A01EventAction.hh"
```

```
#include "A01EventActionMessenger.hh"
#include "G4Event.hh"
#include "G4EventManager.hh"
#include "G4HCofThisEvent.hh"
#include "G4VHitsCollection.hh"
#include "G4TrajectoryContainer.hh"
#include "G4Trajectory.hh"
#include "G4VVisManager.hh"
#include "G4SDManager.hh"
#include "G4UImanager.hh"
#include "G4ios.hh"


A01EventAction::~A01EventAction()
{
;}

void A01EventAction::BeginOfEventAction(const G4Event* evt)
{
  G4int event_id = evt->GetEventID();

  if ((event_id < 100) || (event_id%10000 == 0))
    G4cout << ">>> Event " << evt->GetEventID() << G4endl;
}

void A01EventAction::EndOfEventAction(const G4Event* evt)
{};
// --------------------------------------------------------------
//      GEANT 4 - A01app
//      A01RunAction.c
// --------------------------------------------------------------

#include "A01RunAction.hh"
#include "G4Run.hh"
#include "G4RunManager.hh"
#include "G4UnitsTable.hh"

A01RunAction::A01RunAction()
{;}

A01RunAction::~A01RunAction()
{;}

void A01RunAction::BeginOfRunAction(const G4Run* aRun)
{
  G4cout << "### Run " << aRun->GetRunID() << " start." << G4endl;

}

void A01RunAction::EndOfRunAction(const G4Run* aRun)
{

  G4cout << "\n----------------------End of Run-----------------------------\n";
  G4cout
     << "\n Total hits in position 1 : " << hitsForproton1
     << "\n Total hits in position 2 : " << hitsForproton2
     << "\n Total hits in position 3 : " << hitsForproton3
     << G4endl;
          G4cout << "\n--------------------------------------------------------------\n";
```

```
}
```

APPENDIX B

C PROGRAM FOR DELAYED NEUTRON MEASUREMENT

Delayed neutron measurement program written by Alfred Hanna and Yong Chen

```
/*****************************************************************
* Name:      main.c
*****************************************************************
#include "main.h"
#include "variables.h"

void main()
{
    initialize ();

    bit_num=3;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);

    if (bit_value == 1)                                    // permit
reading
    {
        printf("You NEED the PERMIT to run the system!\n");
        main ();
    }
    else
    {
        clear_screen ();
        sensor_menu ();                                    //Sensor
selection

        ULStat = cbDIn (BoardNum, port_type, &bit_value);

        if (flag_sensor == 1 && bit_value > 0xF7)

            printf("Check the sensors or power -- No sensors selection.\n");

        else if ((flag_sensor == 2 && bit_value > 0xF3))
            printf("Check the sensors or power -- One sensor selection.\n");

        else if (flag_sensor == 3 && bit_value != 0xF3)
            printf("Check the sensors or power -- Two sensors
selection.\n");

        else
        {
            while (1)
            {
                clear_screen ();
                main_menu ();
            }
        }
        exit(0);
    }
 }
```

```
void shoot_solenoid_1 (void)
{
       printf("Solenoid 1 fired!\n\n");
       ULStat = cbDOut(BoardNum =0,AUXPORT,1);
       delay_3s ();
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}


void shoot_solenoid_2 (void)
{
       printf("Solenoid 2 fired!\n\n");
       ULStat = cbDOut(BoardNum =0,AUXPORT,2);
       delay_3s ();
       ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}

void shoot_solenoid_3 (void)
{
       printf("Solenoid 3 fired!\n\n");
       ULStat = cbDOut(BoardNum =0,AUXPORT,4);
       delay_3s ();
       ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}

void close_all_solenoid(void)
{

       ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}

#define BIOS_VIDEO   0x10

void clear_screen (void)
{
    COORD coordOrg = {0, 0};
    DWORD dwWritten = 0;
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    if (INVALID_HANDLE_VALUE != hConsole)
        FillConsoleOutputCharacter(hConsole, ' ', 80 * 50, coordOrg,
&dwWritten);

    MoveCursor(0, 0);

     return;
}

void
MoveCursor (int x, int y)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    if (INVALID_HANDLE_VALUE != hConsole)
    {
```

```
        COORD coordCursor;
        coordCursor.X = (short)x;
        coordCursor.Y = (short)y;
        SetConsoleCursorPosition(hConsole, coordCursor);
    }

    return;
}

void
GetTextCursor (int *x, int *y)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;

    *x = -1;
    *y = -1;
    if (INVALID_HANDLE_VALUE != hConsole)
    {
        GetConsoleScreenBufferInfo(hConsole, &csbi);
        *x = csbi.dwCursorPosition.X;
        *y = csbi.dwCursorPosition.Y;
    }

    return;
}
/*****************************************************************
* Name:      execute_menu.c
*****************************************************************

#include "main.h"
#include "execute_menu.h"

int execute_menu (void)                {
    if (flag == 1)
    {
        flag = 0;
        result_menu ();
    }

    printf("Press 1 and then Enter key to shoot the sample.\n");
    printf("Press 2 and then Enter key to return to main menu.\n");

    for (i=0;i<41;++i)
        selection[i] = 0;
    i=0;

    while ((ch = getchar()) != '\n')
        selection[i++] = ch;

    if (flag == 1)
    {
        flag = 0;
        result_menu ();
```

```
}

if((selection[0] == '1' && selection[1] == 0))                    {
    flag=0;
    printf("Enter irradiation time in seconds --> ");
    scanf("%d", &irradiation_time);
    printf("\nEnter reading time for the\n    detector in seconds ");
    printf("and press\n     Enter to execute the program --> ");
    scanf("%d", &detector_time);
    clear_screen ();
    RegName = LOADREG1;
    ULStat = cbCLoad (BoardNum, RegName, LoadValue);
    core_sample_inside_flag = 0;
    detector_sample_pass_flag = 0;
    core_no_sensor_flag = 0;
    detector_no_sensor_flag = 0;
    loadreg2_flag=0;
    loadreg3_flag=0;
    loadreg4_flag=0;
    core_init_count=0;
    detector_init_count=0;
    core_final_count=0;
    detector_final_count=0;
    flag_core=0;
    detector_sensor_trigger=0;
    two_sensor_execution_flag=0;
    one_sensor_execution_flag=0;
    no_sensor_execution_flag=0;
    min=0;
    sec=0;
    millisec=0;

    while (1)
    {

        CounterNum = 1;
        ULStat = cbCIn (BoardNum, CounterNum, &count);
        if(!(current_value == count))
        {
            master_clock_conversion ();


            if(flag_sensor == 3)            {
                two_sensor_execution ();
                if(two_sensor_execution_flag == 5)
                {
                    break;
                }
            }

            else if(flag_sensor == 2) //  using only detector sensor
            {

                one_sensor_execution ();
```

```
                    if(one_sensor_execution_flag == 5)
                    {
                        break;
                    }

                }

                else if (flag_sensor == 1) // using no sensor
                {
                    no_sensor_execution ();
                    if(no_sensor_execution_flag == 5)
                    {
                        break;
                    }
                }

                else
                {
                    printf("break out!!!!!!\n\n");
                    break;
                }

            }
            current_value = count;

        }
    }

    else if((selection[0] == '2' && selection[1] == 0))
    //return to main_menu
    {
        clear_screen ();
        main_menu ();
    }

    else                            {
        clear_screen ();
        flag=0;
        printf("Value input was incorect, please re-enter the correct
selection.\n\n");
        execute_menu ();
    }
    execute_menu ();
    return 0;
}
/*****************************************************************
*
* Name:      master_clock_conversion.c
*****************************************************************

#include "main.h"
#include "master_clock_conversion.h"

void master_clock_conversion (void)
```

```
{
    MoveCursor (0, 0);
    sec = (count-5536)/1000;
    sec_total = 60*min+sec;
    millisec = count-1000*sec-5536;
     millisec_total = sec_total*1000+millisec;
     sec1 = sec%10;
    sec2 = (sec-sec%10)/10;
    min1 = min%10;
    min2 = (min-min%10)/10;
    millisec1 = (millisec%100)%10;
    millisec2 = ((millisec-millisec1)/100)%10;
    millisec3 = (millisec-millisec1-10*millisec2)/100;

    if (count==65535)
    {
        ++min;
        clear_screen ();
    }
    printf("Master Clock: %d%d:%d%d:%d%d%d\n", min2, min1, sec2, sec1,
millisec3, millisec2, millisec1);
}
/*****************************************************************
*
* Name:      no_sensor_execution.c
*****************************************************************

#include "main.h"
#include "no_sensor_execution.h"

void no_sensor_execution (void)
{

    if (no_sensor_execution_flag == 0)
    {
        no_sensor_execution_flag++;
        printf("Solenoid 1 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,1);   //Solenoid 1 fired
        core_init_count = millisec_total;
    }

    if ((no_sensor_execution_flag == 1) && (core_init_count + 1000 ==
millisec_total))
    {
        no_sensor_execution_flag++;
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        core_init_count = millisec_total;
        flag_core = irradiation_time*1000 + core_init_count;
    }

    if (flag_core == millisec_total && no_sensor_execution_flag == 2)
    {

        no_sensor_execution_flag++;
```

```
        printf("Solenoid 2 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,2);//shoot solenoid 2
        core_final_count = millisec_total;
    }
    if ((no_sensor_execution_flag == 3)&& (core_final_count + 1000 ==
millisec_total))
    {
        no_sensor_execution_flag++;
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        detector_init_count = millisec_total;
        flag_detector = detector_time*1000 + detector_init_count;

    }

    if (flag_detector == millisec_total && no_sensor_execution_flag ==4)
    {
        no_sensor_execution_flag++;
        detector_final_count = millisec_total;
        shoot_solenoid_3 ();

    }
}
/*********************************************************************
*
* Name:      one_sensor_execution.c
*********************************************************************

#include "main.h"
#include "one_sensor_execution.h"

void one_sensor_execution (void)
{

    if (one_sensor_execution_flag == 0)
    {
        one_sensor_execution_flag++;
        printf("Solenoid 1 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,1);   //Solenoid 1 fired
        core_init_count = millisec_total;
    }

    if ((one_sensor_execution_flag == 1) && (core_init_count + 1000 ==
millisec_total))
    {
        one_sensor_execution_flag++;
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        core_init_count = millisec_total;
        flag_core = irradiation_time*1000 + core_init_count;
    }

    if ((flag_core == millisec_total) && (one_sensor_execution_flag == 2))
    {
        one_sensor_execution_flag++;
        printf("Solenoid 2 fired!\n\n");
```

```
        ULStat = cbDOut(BoardNum =0,AUXPORT,2);//shoot solenoid 2
        core_final_count = millisec_total;
    }

    bit_num=2;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
    detector_sensor_trigger = bit_value;

    if (detector_sensor_trigger == 1 && detector_sample_pass_flag == 0 &&
one_sensor_execution_flag == 3)
    {
        one_sensor_execution_flag++;
        detector_sample_pass_flag++;
        detector_init_count = millisec_total;  // Timestamp - record time
sample arrived in the detector
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        flag_detector = detector_time*1000 + detector_init_count;
    }
    if (flag_detector == millisec_total && one_sensor_execution_flag == 4)
    {
        one_sensor_execution_flag++;
         detector_final_count = millisec_total;   // Timestamp - record
time sample left the detector
        shoot_solenoid_3 ();
    }


}
/*********************************************************************
*
* Name:      two_sensor_execution.c
*********************************************************************

#include "main.h"
#include "two_sensor_execution.h"

void two_sensor_execution (void)
{
    if (two_sensor_execution_flag == 0)
    {
        two_sensor_execution_flag++;
        printf("Solenoid 1 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,1);   //Solenoid 1 fired
    }

    // Read the value off the sensor in the core

    bit_num=1;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
    core_sensor_trigger = bit_value;

    if (two_sensor_execution_flag == 1 && core_sensor_trigger==0)
    {
        two_sensor_execution_flag++;
```

```
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        core_init_count = millisec_total;
        flag_core = irradiation_time*1000 + core_init_count;
    }

    if ((flag_core == millisec_total) && (two_sensor_execution_flag == 2))
    {
        two_sensor_execution_flag++;
        core_final_count = millisec_total; // Timestamp - record time
sample left the core

        printf("Solenoid 2 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,2);
    }

    // Read the value off the sensor in the detector

    bit_num=2;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
    detector_sensor_trigger = bit_value;

    if (detector_sensor_trigger == 1 && detector_sample_pass_flag == 0 &&
two_sensor_execution_flag ==3)
    {
        two_sensor_execution_flag++;
        detector_sample_pass_flag++;
        detector_init_count = millisec_total;  // Timestamp - record time
sample arrived in the detector
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        flag_detector = detector_time*1000 + detector_init_count;
    }
    if (flag_detector == millisec_total && two_sensor_execution_flag == 4)
    {
        two_sensor_execution_flag++;
         shoot_solenoid_3 ();
        detector_final_count = millisec_total; // Timestamp - record time
sample left the detector
    }
}

/*****************************************************************
*
* Name:      test_menu
*****************************************************************

#include "main.h"

extern char selection[40];
extern char ch;                 // character
extern int i;               // position in the array

void test_menu (void)                   //main_menu --> Selection 2 -
Testing the solenoids
{
```

```
    // Initializes Solenoids to be closed

    close_all_solenoid ();

    printf("Press 1 and then Enter key to activate Solenoid 1.\n");
    printf("Press 2 and then Enter key to activate Solenoid 2.\n");
    printf("Press 3 and then Enter key to activate Solenoid 3.\n");
    printf("Press 4 and then Enter key to return to main menu.\n");

    for (i=0;i<41;++i)
        selection[i] = 0;
    i=0;

    while ((ch = getchar()) != '\n')
        selection[i++] = ch;

    if(selection[0] == '1' && selection[1] == 0)  //shoots sample -->
Dout1 using decimal value 1
    {
        clear_screen ();
        shoot_solenoid_1 ();
        test_menu ();
    }
else if(selection[0] == '2' && selection[1] == 0)
    {
        clear_screen ();
        shoot_solenoid_2 ();
        test_menu ();
    }
else if(selection[0] == '3' && selection[1] == 0)
    {
        clear_screen ();
        shoot_solenoid_3 ();
        test_menu ();
    }
    else if(selection[0] == '4' && selection[1] == 0)
    {
        clear_screen ();
        main_menu ();
    }
    else                            //wrong selection, re-enter the
selection
    {
        clear_screen ();
        printf("Value input was incorect, please re-enter the correct
selection.\n\n");
        test_menu ();
    }
}
```

VITA

Yong Chen received his B.S. in nuclear engineering from Tsinghua University, Beijing, China on July 1, 2000 and his M.S. in radiation protection from Tsinghua University on July 1, 2002.

Mr. Yong Chen can be reached at 854 Brookwood Dr Apt 201 Oklahoma City OK 73139 and his permanent email address is radoncy@gmail.com.