

AD-A253 917



DTIC
S **ELECTE** **D**
JUL 15 1992
C

(2)

PL-TR-92-2084

**AN ACCURATE AND EFFICIENT METHOD TO CALCULATE THE
RATE OF SINGLE EVENT UPSETS FROM THE LET SPECTRUM
AND SEU CROSS SECTION TEST RESULTS**

**D. L. Chenette
T. L. Schumaker
A. E. Williamson**

**Lockheed Palo Alto Research Laboratory
Space Sciences Laboratory 0/91-20, B/255
Palo Alto, CA 94304-1191**

20 March 1992

Scientific Report No. 1

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED



**PHILLIPS LABORATORY
AIR FORCE SYSTEMS COMMAND
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731-5000**

92-18394



92 7 13 128

"This technical report has been reviewed and is approved for publication"


KEVIN P. RAY, LT, USMC
Contract Manager


E. G. MULLEN
Branch Chief


WILLIAM SWIDER
Deputy Director

This document has been reviewed by the ESD Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS).

Qualified requestors may obtain additional copies from the Defense Technical Information Center. All others should apply to the National Technical Information Service.

If your address has changed, or if you wish to be removed from the mailing list, or if the addressee is no longer employed by your organization, please notify PL/TSI, Hanscom AFB, MA 01731. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 3-20/92		3. REPORT TYPE AND DATES COVERED Scientific #1	
4. TITLE AND SUBTITLE AN ACCURATE AND EFFICIENT METHOD TO CALCULATE THE RATE OF SINGLE EVENT UPSETS FROM THE LET SPECTRUM AND SEU CROSS SECTION TEST RESULTS				5. FUNDING NUMBERS F19628-90-C-0101 PE 62101F PR 7601 TA22 WULB	
6. AUTHOR(S) D. L. Chenette, T. L. Schumaker, and A. E. Williamson					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lockheed Palo Alto Research Laboratory Space Sciences Laboratory, 0/91-20 B/255 3251 Hanover Street Palo Alto, CA 94304-1191				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Phillips Laboratory Hanscom AFB, MA 01731-5000 Contract Manager: Kevin P. Ray, Lt., USAF/GPSP				10. SPONSORING/MONITORING AGENCY REPORT NUMBER PL-TR-92-2084	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; Distribution unlimited				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) A specific implementation is presented for calculating the rate of single event upsets in microelectronic devices. The implementation is based on the method of Pickel and Blandford (1980), who combined device test results, the flux distribution of particles in the space environment, and a model for the geometry of the device. The device test data are summarized as an upset cross section expressed as a function of the linear energy transfer (LET). The flux distribution of particles in the environment is described by an integral LET spectrum. The device geometry model is that of a solid rectangular prism. A numerical integration of the LET spectrum and path length distribution in the solid over the upset cross section is performed to obtain the resultant upset rate.					
14. SUBJECT TERMS Single event upset, radiation effects, microelectronics				15. NUMBER OF PAGES 42	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT SAR		

Contents

Introduction	1
Methods and procedures	1
LET spectrum	1
Path length distribution	2
Device geometry	2
Total upset rate	3
Integrating detailed upset cross sections	3
Application of the method	6
References	8
Appendix A: SEU_CALC and related subroutines	9
Source code organization	9
Subroutine descriptions	10
Input and output file descriptions	12
Appendix B: Sample LET_SPECTRUM input file	15
Appendix C: Sample DEVICE input file	17
Appendix D: Sample SEU_CALC output file	19
Appendix E: Source code listings	21



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

List of Figures and Tables

Figure 1: Measured single event upset cross sections for the 93L422 RAM

Figure 2: Step function approximations to the upset cross section

List of Symbols, Abbreviations, and Acronyms

AMD	Advanced Micro Devices
CRRES	Combined Release and Radiation Effects Satellite
LET	Linear energy transfer
RAM	Random access memory
SEU	Single event upset

Introduction

The goal of the effort reported here is to provide a convenient, efficient, and accurate method to estimate the rate of single event upsets for devices in the space environment during the various phases of the CRRES mission. This calculation integrates results from the CRRES cosmic ray model, results from ground tests of device upset susceptibility, approximate device geometry models, and device shielding estimates to obtain the upset rate. The method provides upper and lower limit estimates of the upset rate from different integrations of the upset cross-section test results. These limits provide some indication of the precision of the result and its sensitivity to the quality of the device test data.

This report provides an overall description of the approach, a detailed description of the specific method adopted, and full documentation of the software implementation of that method.

Methods and Procedures

The method chosen to calculate the SEU rate is similar to that described by Pickel and Blandford (1980). It is a direct application of a few simple geometrical concepts.

The upset rate is determined as the rate at which energetic heavy ions in the space radiation environment can deposit more than a certain minimum amount of energy into a specific volume. The minimum energy deposit required to cause an upset is called the upset threshold. The specific volume is the sensitive volume of a cell in the device. The particle rate is determined from the product of the local particle flux (per unit time per unit solid angle and per unit area) and the geometrical factor presented by the sensitive volume (solid angle, area product).

The requirement to deposit more than a certain minimum amount of energy into a specific volume restricts the calculation. It leads to a specific method to organize the particle flux distribution and a specific method to describe the geometry of the sensitive region. This modified description of the particle flux distribution is called the linear energy transfer (LET) spectrum. The modified description of the geometry of the sensitive region is called the path length distribution.

LET spectrum The linear energy transfer (LET) of an energetic heavy ion, the energy loss per unit distance travelled in a material, generally increases with the atomic number of the ion (Z) and generally decreases with increasing velocity (except at low and high velocities where atomic effects and relativistic effects, respectively, become important). Thus a fast iron ion and a slower oxygen ion can have the same LET and, by assumption, the same ability to

produce a single event upset in a specific device. The LET spectrum is a convenient way to integrate and keep track of the contributions of various ions ordered according to LET. The integral LET spectrum used here is reported as a product of the CRRES Cosmic Ray Model Group. It represents the total flux of particles of all types with an LET larger than some threshold (L). This is written as $F(L)$.

Path length distribution The energy deposited by a particle with LET L_0 along path l is $E_D = L_0 \cdot l$. Thus the other information needed to establish the energy deposited by an ion penetrating a region is the length of the chord through the region (l) representing the track of the ion.

Viewed from an arbitrary direction, any solid presents a distribution of chord lengths or projected thicknesses, over its projected area. This is the unidirectional path length distribution through the solid. It has a simple interpretation but can be rather complicated to express mathematically.

For any solid at an arbitrary orientation, there is an area, solid angle product (geometrical factor) presented by the solid for a given path length. In the present work, since the region of interest has an arbitrary orientation in space, the path length distribution is computed as the average over all orientations. The path length distribution is written $P(l)$. It is zero for path lengths longer than the maximum chord through the volume (l_{max}). For the work reported here, the distribution is normalized so that the integral over all path lengths is equal to the total geometrical factor of the solid. For many types of solid prisms this is π times the surface area of each face of the solid. For a right rectangular solid prism with dimensions x, y, z the path length distribution is written as $P(l; x, y, z)$ and the normalization condition is:

$$\int_0^{l_{max}} P(l; x, y, z) dl = 2\pi(xy + xz + yz) \quad (1)$$

The maximum path length is given by:

$$l_{max} = \sqrt{x^2 + y^2 + z^2} \quad (2)$$

The form of the function $P()$ is provided by Pickel and Blandford (1980), although they choose to use a different normalization than presented here.

Device geometry Single-event upset susceptibility tests of microelectronic devices are often characterized by two parameters: a cross-sectional area of the device which is sensitive to upset, and a threshold value of the LET necessary to induce an upset (ref.) In the simplest idealization of the test results, the upset cross-section is zero below some threshold LET value, L_C , and a constant equal to some area, A_U , above this threshold.

The upset cross-section results yield to a simple geometric interpretation which has been validated by continuing device tests and analyses. The sensitive region of any cell, or bit of RAM, is often modelled as a rectangular solid section of the silicon chip. This section has a thickness (z) and some

surface area ($x \cdot y$). The geometric interpretation of the total sensitive area measured in the SEU tests is clear: For a device with N cells (e.g. N bits in a RAM) the total sensitive area of the device is $A_U = N \cdot x \cdot y$. Thus, a measurement of the upset cross section together with a knowledge of the number of bits (N) yields an estimate of the sensitive cell area ($x \cdot y$). The individual dimensions (x and y) can often be obtained with sufficient accuracy just as the square root of that area. More precise estimates are rarely required or justified. They must come from details of the manufacture of the device or from physical analysis of the structure.

The upset threshold value of the LET (L_C) multiplied by the thickness of the region (z) is interpreted as a minimum energy or charge deposit necessary to change the state of the device, $E_C = z \cdot L_C$. (For silicon, this energy deposit or charge deposit are proportional to each other. The constant of proportionality is 3.6 eV/electron.) The thickness (z) is not generally known with great precision. It depends on parameters associated with the device technology and wafer processing. Fortunately, since both the test results and the real spectrum are expressed in terms of the LET, an imprecise knowledge of the depth (z) has little effect on the precision of the result. This has been demonstrated by detailed calculation (Shoga, et al., 1988).

Total upset rate The total upset rate (U) for the simple model outlined above is given by the following expression:

$$U = \int_0^{\infty} F(E_C / l) P(l; x, y, z) dl \quad (3)$$

The integral is performed over the entire path length distribution. However, for any path length, l , only particles with LET above the threshold $L = E_C / l$ contribute to the upset rate. Because the LET spectrum and the path length distribution are not simple analytic functions, the integral is generally computed numerically.

Integrating Detailed Upset Cross-Sections

The relatively simple integral described in the previous section is appropriate in the idealized situation where the measured upset cross section is a step function, equal to zero below some threshold value of the LET and equal to a constant value above it. The situation becomes slightly more complicated in the real world. In this section other shapes for the upset cross-section are addressed.

The upset cross-section measurements are rarely step-functions. More often, there is a transition region of LET where the cross-section rises from zero to some asymptotic value (Figure 1). This behavior has been discussed extensively in the literature and various explanations offered. For our purposes, however, we choose to adopt another simple concept to understand this effect.

This concept also motivates and justifies our method to incorporate the effect into the calculation.

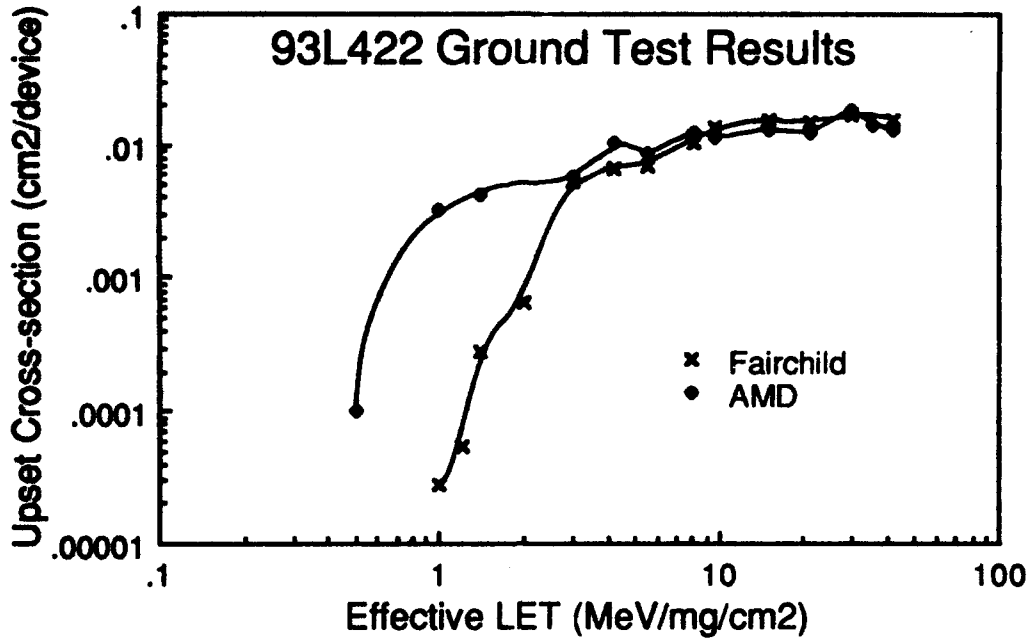


Figure 1: Measured single event upset cross sections for the 93L422 RAM

The concept to apply to the variation in upset cross-section with LET is to treat the ratio of the upset cross-section to its asymptotic value as an upset "effectiveness" or "efficiency". To accomplish this the upset cross section is divided by the area factor ($x \cdot y$). At low values of the LET the efficiency for upset in the sensitive region is low. As the LET increases, the efficiency also increases. As the LET is increased still more an asymptotic cross section is established which defines the area sensitive to the SEU process. The upset cross-section divided by its asymptotic value is an efficiency in the range from 0 to 1. The area factor ($x \cdot y$) is retained through the path length distribution normalization as in the discussion above.

In applying this concept to the integral necessary to calculate the SEU rate, the cross-section, renormalized as an efficiency, can be treated as a sum of step functions in the LET ($h(L)$) with amplitude s_i at location L_i :

$$\varepsilon(L) = \frac{\sigma(L)}{x \cdot y} = \sum_i [s_i h(L_i) - s_{i-1} h(L_i)] \quad (4)$$

The subtraction is necessary to remove the contribution of one step function from the region to the right of the next. All of the values s_i are in the range from 0 to 1 and monotonically increasing with increasing L_i . The sequence of values s_i and L_i can be chosen in many ways to approximate the shape of the upset cross

section. Lower and upper limit estimates can be chosen and refined at will. An example is illustrated in Figure 2.

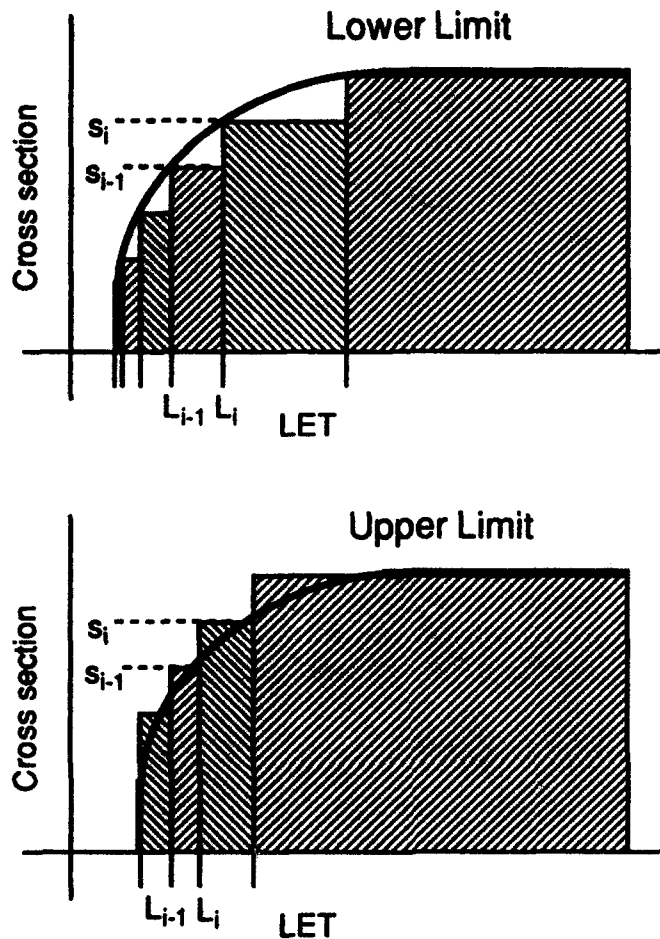


Figure 2: Step function approximations to the upset cross section

With this formulation of the measured upset cross section, the complete formula for the upset rate becomes:

$$U = \sum_i (s_i - s_{i-1}) \int_0^{l_{max}} F(z L_i / l) P(l; x, y, z) dl \quad (5)$$

The factor $z \cdot L_i$ is the critical energy deposit threshold for each step function piece. This completes the presentation of the method.

Application of the method

Software to implement the method described in the previous section has been developed. A detailed description of the subroutines and listings of the source code are presented in the appendices. Some practical considerations for the use of this software are provided in this section.

Following the discussion above, there are four major components in the calculation. Each of these components must be provided. Most of them are input parameters or input data files. The major components are:

1. An estimate of the dimensions of the device cell which is sensitive to upset, modelled as a rectangular solid prism, and the number of sensitive regions in the device. The dimensions are labelled x , y , and z . By convention, the z dimension is normal to the face of the chip die. The area $A = x*y$ is the maximum total sensitive area of a unit cell (the sensitive area corresponding to a bit in a RAM, for example). N is the number of unit cells in the device. (The number of bits, for example.) The quantity $N*x*y$ should be equal to the asymptotic value of the upset cross section (part 3, below).

2. The distribution of path lengths through the sensitive region, $P(l)$. The model provided in the code is for that of a solid rectangular prism.

3. The upset cross-section of the device as a function of LET, $\sigma(L)$. The cross-section is obtained from ground tests. Physically, to be consistent with the assumptions of the model, the upset cross section must be monotonically increasing with increasing LET. Often the test data are not monotonic (c.f. Figure 2). This is attributed to imperfections in the test results and the scatter around a smooth monotonic curve can be used as an estimate of the standard deviation of the test data set. The data set which is to be used in this software must be smoothed (by hand, if necessary) so that it is monotonic.

4. The flux of particles in the radiation environment as a function of LET, $F(L)$. This distribution is provided as one of the products of the CRRES Cosmic Ray Model, calculated from models and/or measurements of the abundances and energy spectra of the energetic heavy ions in the cosmic rays. It is calculated for a variety of assumptions for the thickness of shielding between the environment and the sensitive region of the device. The LET spectrum for the proper epoch and for the proper thickness of passive shielding must be selected to obtain accurate upset rate estimates.

For each run, the upset rate is calculated three times, once each for a lower limit, an upper limit, and a central estimate. All of these estimates are controlled through the input cross section data file. Both upper and lower limits are defined with respect to the LET and cross section values provided in the file. In this calculation of the SEU rate, the step functions used for the lower and upper limit estimates are placed at the values of the LET which are provided in the input file. The lower limit model uses the cross section listed in the file at each LET. This yields a lower limit estimate as indicated by inspection of Figure 2. The upper limit model uses the cross section provided in the input file at the

next higher value of the LET. This yields an upper limit estimate, again, as is clear from inspection of Figure 2. The central estimate is obtained by a 20-point linear interpolation of the listed cross section vs LET values. The effects of various uncertainties in the measured cross sections can be investigated by modifying the cross section data file. Different runs can be made with fewer entries in the cross section table or with the additional cross section values obtained by any of a variety of user-generated interpolation schemes.

Additional details of specific data formats and the mechanics of using the code are given in the appendices.

References

- Kolasinski, W.A., R. Koga, and D.L. Chenette, "Heavy ion induced single event upsets in a bipolar logic device", *IEEE Transactions on Nuclear Science*, NS-30 4470, 1983.
- Pickel, James C. and James T. Blandford, Jr., "Cosmic-ray-induced errors in MOS devices", *IEEE Transactions on Nuclear Science*, NS-27, 1006, 1980.
- Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, UK, 133, 1988.
- Shoga, M., P. Adams, D.L. Chenette, R. Koga, and E.C. Smith, "Verification of single event upset rate estimation methods with on-orbit observations", *IEEE Transactions on Nuclear Science*, NS-34, 1256, 1987.

Appendix A

CRRES Cosmic Ray Model Single Event Upset Rate Calculator

SEU_CALC and related support subroutines

Source code organization and layout

All of the source code is written in ANSI C. The source code is contained in three files named SEU_CALC.C, PATHDIST.C, and INP_LIST.C. All of the source code is in the public domain. The subroutines contained in each file are listed below:

SEU_CALC.C
 main()
 integrand()
 myflux()
 lgauss()
 do_int()

PATHDIST.C
 pldset()
 pldt()
 pld()

INP_LIST.C
 input_list()

Subroutine descriptions

A brief description of the purpose served by each subroutine is provided here together with the relationships between them and calling sequences.

Subroutine name: main()

Calling sequence: None, entry point for SEU_CALC

Returned value: None

Description: main() is the entry point for the SEU_CALC package. It opens the output file, handles the user interactions, writes the documentation of the run, handles units conversions, and calls the other routines in order as required.

Subroutine name: integrand()

Calling sequence: double integrand(double l), where l is a path length in microns through the sensitive volume of the device

Returned value: a double-precision value of the integrand for path length l given the path length distribution, the LET spectrum, and a critical charge threshold

Description: Given a path length distribution $p(l)$, a critical charge threshold Q , and an LET spectrum $F(L)$, integrand(l) returns the number of particles per second capable of depositing more than Q in path length l as $p(l)*F(L=Q/l)$

Subroutine name: myflux()

Calling sequence: double myflux(double L), where L is an LET threshold in units of picocoulombs per micron

Returned value: a double-precision value of the integral flux with LET greater than L $F(L)$

Description: Given the tabulation of the LET spectrum, myflux(L) returns the interpolated value of the spectrum at $LET=L$. If $L<0$ or L is greater than the largest value in the tabulation, the flux returned is zero.

Subroutine name: lgauss()

Calling sequence: double lgauss(double a, double b)

Returned value: a double-precision value of the integral of integrand() (above) from a to b

Description: lgauss does a ten-point Gauss-Legendre integration based on Press, *et al.* (1988).

Subroutine name: do_int()

Calling sequence: int do_int(double Acc, double min, double max, double *res, double *old)

Returned value: an integer number of integration subdivisions used as well as the current best (*res) and previous (*old) integrals

Description: do_int() integrates the value of integrand() (above) from min to max. The number of subdivisions of the integration is

increased until two successive results (a_1 and a_2) agree to within less than $Acc \cdot a_2$. The integral is first performed with a 20-point trapezoidal rule. Then it is repeated using Gauss-Legendre integrals over smaller and smaller divisions of the range of integration until convergence to the required accuracy is achieved.

Subroutine name: `pldset()`

Calling sequence: `double pldset(double x, double y, double z)`

Returned value: a double value number equal to the maximum path length through the sensitive region

Description: `pldset()` tabulates the path length distribution for a right rectangular solid prism as calculated by `pld()`. This is done to speed up the calculation. The path length distribution is tabulated at 1000 points between the minimum and maximum. When values are required later they can be obtained by simple interpolation.

Subroutine name: `pldt()`

Calling sequence: `double pldt(double s)`

Returned value: the double value of the path length distribution at path length s

Description: `pldt()` interpolates the tabulation made by `pldset()`

Subroutine name: `pld()`

Calling sequence: `double pld(double s, double x, double y, double z)`

Returned value: the double value of the path length distribution at path length s for a right rectangular solid of dimension x, y, z

Description: `pld()` implements the formula in the appendix of Pickel and Blandford (1980). `pld()` uses a different normalization than do Pickel and Blandford, however. In `pld()` the path length distribution is normalized such that the result returned is the geometrical factor of the right rectangular solid prism for chords of length s . The units of this result are area * steradians with the area as the square of the units in which x, y, z , and s are presented. The integral of `pld()` over all values of s is equal to the total geometry factor of the solid, which is equal to π times the sum of the areas of the six faces.

Subroutine name: `input_list()`

Calling sequence: `int input_list(char *file, char *key, int max, double *c1, double *c2, char *title)`

Returned value: the number of items retrieved from the file

Description: `input_list()` reads the key-based input data files needed by the program. The file name is contained in the `*file` string and the selection key in the `*key` string. The `*title` string is used to return the comment following the key (if `maxn` is non-zero). If `maxn` is zero then the value returned in `*c1` is the numerical value immediately following the key. Otherwise `max` defines the maximum number of pairs of entries to be returned from the table. `c1` and `c2` should point to double arrays that are at least `max` long.

The values read from the table columns in the input file are returned in c1, for the first column, and in c2 for the second column.

Input and output file descriptions

Two files of input data are required to perform the SEU rate calculation: an LET spectrum file (LET_SPECTRUM), and a device cross-section file (DEVICE). Both files are text files with the necessary information presented following key strings or in a tabular, columnar format. The contents and the format requirements and restrictions for these files are described here. In addition to writing to the user terminal or console screen, the program also writes its output to a file to provide a permanent record of the run.

Input file format rules and restrictions:

Input files are ASCII files containing descriptive text, a key string of characters, and numbers. Lines are delimited by either a new-line character or by an end of file. The values may be in any floating-point format recognized by the ANSI C atof() function. The values found in the input files as interpreted by the program are written to the output file. When first using a new input DEVICE or LET_SPECTRUM file, the output should be carefully checked to ensure that the interpretation of the data is correct and complete.

LET spectrum file:

This file must be named or (in VMS, for example) defined or assigned to the name LET_SPECTRUM. The key string is the three-character string "LET". Any number of lines of text may begin the file as long as the sequence "LET" is not present. The first line containing "LET" is assumed to be a comment line and is copied to the output as partial documentation of the run. Following this comment line must be a series of lines containing two values per line as in a two-column tabular form. This series of values is the tabulation of the LET spectrum. The first value on each line must be the LET in units of MeV/micron. The second value must be the integral flux corresponding to that value of the LET in units of particles per square meter per steradian per second. A maximum of 200 entries in the LET spectrum table is supported at present. This number can be changed by changing the value of NUMSPC and re-compiling. A sample LET spectrum file is provided as Appendix B.

Device upset cross-section file:

This file must be named or (in VMS, for example) defined or assigned to the name DEVICE. There are five sets of key strings which must appear in the file. The first four key strings are the two-character sequences: "X=", "Y=", "Z=", and "N=". The last key is the three-character string "LET". The "X=", "Y=", "Z=", and "N=" key strings must each be followed immediately (except for optional

blank spaces and tabs) by a numeric value. This value is interpreted and assigned to the respective variable.

The first three keys describe the physical characteristics of the device and its sensitive volume. X, Y, and Z are the dimensions of the sensitive region cell in microns. The Z dimension is assumed to correspond to depth in the silicon. The quantity $X*Y$ is the sensitive surface area of the cell.

The number of sensitive volumes in the device cross-section table is defined by the value of N. If the cross-section is tabulated per bit, N should be set to 1.

Any number of lines of text may begin the file (these should include the "X=", "Y=", "Z=", and "N=" key strings) as long as the sequence "LET" is not present. The first line containing "LET" is assumed to be a comment line and is copied to the output as partial documentation of the run. Following this comment line must be a series of lines containing two values per line as in a two-column tabular form. This is the tabulation of the device upset cross-section. The first value on each line must be the LET in units of MeV per milligram per square centimeter. The second value must be the device upset cross-section corresponding to that value of the LET in units of square centimeters per device (or square centimeters per bit if the number of bits is set to 1, above). A maximum of 50 entries in the device upset cross-section table is supported at present. This number can be changed by changing the value of NUMXST and re-compiling. A sample device upset cross-section file is provided as Appendix C.

Output file description:

One output file is generated whenever SEU_CALC is run. It is named SEU_CALC. The output file provides a version number of the program, the user-selected convergence parameter, documentation of all of the data used in the calculation as read from the LET_SPECTRUM and DEVICE input files, and the results of the integration. A sample output file is provided as Appendix D, for a run using the input files of Appendices B and C.

Appendix B: Sample LET_SPECTRUM input file

LET (MeV/micron)	Flux (/m2 s sr)	"LEASAT 1" spectrum
{ 1.0000e-02	, 2.1972e+01	},
{ 1.3388e-02	, 1.8184e+01	},
{ 1.7925e-02	, 1.4426e+01	},
{ 2.3998e-02	, 1.1121e+01	},
{ 3.2129e-02	, 8.0889e+00	},
{ 4.3015e-02	, 5.9511e+00	},
{ 5.7590e-02	, 4.3873e+00	},
{ 7.7103e-02	, 3.1657e+00	},
{ 1.0323e-01	, 2.2608e+00	},
{ 1.3820e-01	, 1.6468e+00	},
{ 1.8503e-01	, 1.1756e+00	},
{ 2.4773e-01	, 7.7379e-01	},
{ 3.3166e-01	, 4.0817e-01	},
{ 4.4404e-01	, 1.9998e-01	},
{ 5.9449e-01	, 1.0080e-01	},
{ 7.9592e-01	, 5.1504e-02	},
{ 1.0656e+00	, 2.6613e-02	},
{ 1.4267e+00	, 1.3688e-02	},
{ 1.9100e+00	, 6.8406e-03	},
{ 2.5572e+00	, 3.4166e-03	},
{ 3.4237e+00	, 1.5384e-03	},
{ 4.5837e+00	, 6.7240e-04	},
{ 6.1368e+00	, 1.3942e-04	},
{ 8.2161e+00	, 5.6336e-08	},
{ 1.1000e+01	, 0.0	}

Appendix C: Sample DEVICE input file

This is for the 93L422 RAM

X=39 Y=39 Z=1

N= 1024

/* Upset cross-section data table.
The example given here is the measured
upset cross section for an AMD 93L422 RAM
tested 22 May 1987 at LBL 88"cyclotron.

LET (MeV/mg/cm2)	Cross-Section (cm2/device)
{ 0.1 ,	0.0 },
{ 0.5 ,	1.00e-4 },
{ 1.0 ,	3.23e-3 },
{ 1.4 ,	4.34e-3 },
{ 3.0 ,	5.77e-3 },
{ 4.2 ,	7.20e-3 },
{ 5.5 ,	8.73e-3 },
{ 8.2 ,	1.00e-2 },
{ 9.6 ,	1.12e-2 },
{ 15. ,	1.18e-2 },
{ 21. ,	1.24e-2 },
{ 30. ,	1.30e-2 },
{ 35. ,	1.39e-2 },
{ 42. ,	1.48e-2 }

};

Appendix D: Sample SEU_CALC output file

The input files used in this run were those listed in Appendix B and C. The user selected the "low" convergence condition by entering "1" at the prompt. The output from the SEU_CALC output file follows. To the screen the program writes some additional intermediate results. One purpose of this is to help assure the user that the program is still executing.

SEU rate calculator - 1-D integration method
Integral over experimental cross-section data.
Version 1.0 D.L. Chenette 8 November 1991

Convergence parameter = 1. Integration consistency = 0.1

LET spectrum used in this run:

LET (MeV/micron) Flux (/m2 s sr) "LEASAT 1" spectrum

LET (MeV/micron)	Flux (/m2 s sr)
1.0000e-02	2.1972e+01
1.3388e-02	1.8184e+01
1.7925e-02	1.4426e+01
2.3998e-02	1.1121e+01
3.2129e-02	8.0889e+00
4.3015e-02	5.9511e+00
5.7590e-02	4.3873e+00
7.7103e-02	3.1657e+00
1.0323e-01	2.2608e+00
1.3820e-01	1.6468e+00
1.8503e-01	1.1756e+00
2.4773e-01	7.7379e-01
3.3166e-01	4.0817e-01
4.4404e-01	1.9998e-01
5.9449e-01	1.0080e-01
7.9592e-01	5.1504e-02
1.0656e+00	2.6613e-02
1.4267e+00	1.3688e-02
1.9100e+00	6.8406e-03
2.5572e+00	3.4166e-03
3.4237e+00	1.5384e-03
4.5837e+00	6.7240e-04
6.1368e+00	1.3942e-04
8.2161e+00	5.6336e-08
1.1000e+01	0.0000e+00

LET spectrum set. 25 values tabulated

Device Upset Cross-Section Tabulation

LET (MeV/mg/cm2) Cross-Section (cm2/device)

*/

LET (MeV/mg/cm2)	Cross-section (cm2/device)
1.0000e-01	0.0000e+00
5.0000e-01	1.0000e-04
1.0000e+00	3.2300e-03
1.4000e+00	4.3400e-03
3.0000e+00	5.7700e-03
4.2000e+00	7.2000e-03
5.5000e+00	8.7300e-03
8.2000e+00	1.0000e-02
9.6000e+00	1.1200e-02
1.5000e+01	1.1800e-02
2.1000e+01	1.2400e-02
3.0000e+01	1.3000e-02
3.5000e+01	1.3900e-02
4.2000e+01	1.4800e-02

Sensitive region (microns): X = 39.00 Y = 39.00 Z = 1.00

Number of bits per device = 1024

Cross section set. 14 values tabulated

Integration complete.

Result = 9.3295e-04 upsets per bit per day

Extreme limit bounds are 6.4874e-04 to 1.3410e-03

Result = 9.5534e-01 upsets per device per day

Extreme limit bounds are 6.6431e-01 to 1.3732e+00

Appendix E: Source code listings

Listings of the current version of the source code are included here. All of the source code is written in ANSI C. The source code is contained in three files named SEU_CALC.C, PATHDIST.C, and INP_LIST.C. All of the source code is in the public domain. The subroutines contained in each file are:

SEU_CALC.C
main()
integrand()

myflux()
lgauss()
do_int()

PATHDIST.C
pldset()
pldt()
pld()

INP_LIST.C
input_list()


```
#include <stdio.h>
#include <math.h>
/* Routines to do 1D integral of single-event upset cross-section
   over pathlength distribution and LET spectrum to calculate the
   rate of single-event-upsets.
```

Written by: D.L. Chenette Version 1.3 8 November 1991

Required source code modules: SEU_CALC.C INP_LIST.C PATHDIST.C

SEU_CALC: this module with main, LET spectrum, and cross section models
INP_LIST: input module to read DEVICE and LET_SPECTRUM files
PATHDIST: module to calculate pathlength distribution of rectangular solid

To compile and produce an executable file under VMS, do the following
(make sure all files are in the same directory before you start):

Define the proper link libraries (may be done in the login.com):

DEFINE LINK\$LIBRARY SYS\$LIBRARY:VAXCCURSE.OLB

DEFINE LINK\$LIBRARY_1 SYS\$LIBRARY:VAXCRT.LB

Compile all modules individually with the following commands to
generate .OBJ files (unnecessary if .OBJ file exists already):

CC PATHDIST

CC INP_LIST

CC SEU_CALC

When this is done you should have .OBJ files of each of these.

Link all modules to make the executable file with the following:

LINK SEU_CALC,INP_LIST,PATHDIST

The link step should produce the executable file SEU_CALC.EXE
which you can run with the command RUN SEU_CALC

Before RUNNING, however, you must DEFINE DEVICE and LET_SPECTRUM
input files as well as the SEU_CALC output file. Alternatively,
these file names may be used directly if the files are in the
current directory.

The DEVICE and LET_SPECTRUM input files are nearly free-format.
In each file special text keys are used to distinguish comments
or other text from the information required by the program.

The DEVICE file describes the integrated circuit part for the
SEU rate calculation. The following information is required:

1. Dimensions of the sensitive cell in microns (X,Y,Z).
The Z dimension is assumed to be depth into the silicon.
The quantity X*Y is the sensitive surface area of a cell.
2. The number of sensitive volumes (N) in the cross-section
table which follows. If the cross-section is tabulated
per bit, foreexample, N should be set to 1.
3. A 2-column listing of LET and upset cross-section values.
The units for the LET must be MeV/micron, and the units
for the upset cross-section must be in square centimeters
per device (or per bit, if N is set to 1, see above)

The keys for X, Y, Z, and N are the two-character sets X= Y= Z= or

N= The first numerical value found after the key on the same line is interpreted as the value for the required parameter. The keys and their values may appear on one or more lines. They should appear in the DEVICE file before the cross-section key text string.

For the cross-section table of the DEVICE file and for the LET spectrum of the LET_SPECTRUM file the key is the 3-characters LET. The first line containing "LET" is assumed to be a comment and is listed to the output. Subsequent lines to the end of the file are expected to contain at least 2 numbers (and may contain additional values or other text). The first 2 numbers on each line are used and interpreted as the LET and the upset cross-section (for the DEVICE file) or the LET and the integral flux (per square meter per second per steradian, for the LET_SPECTRUM file). If only one number is found on a line, that line will be ignored as a comment. The format of the numbers not critical. Fixed-point, floating point and exponential notations are recognized. The program lists what it has read to the output file. When using a new input DEVICE or LET_SPECTRUM file this listing should be checked to make sure that it is being interpreted correctly.

Restrictions: The first entry of the DEVICE file should have a non-zero LET with a cross-section of zero to start the integration. The both files must be monotonic increasing in LET. The device file must be monotonic increasing in cross-section with increasing LET. The LET_SPECTRUM file (since it is an integral flux) must be monotonic decreasing in the flux with increasing LET. Maximum number of entries in upset cross-section is now set to 50. Maximum number of entries in the LET spectrum is now set to 200.

If you want to specify a step-function upset cross-section, put in a two line cross-section listing. An example DEVICE file follows:

```
X= 40 Y=40 Z= 1 Sensitive volume is 40x40x1 microns
N= 1024bits LET for step function example for 1K device
1.0 0.0 step function upset cross-section with critical
1.0 0.001 LET threshold = 1 MeV/mg/sqcm and total upset
cross section for the device of 0.001 sqcm
```

User input: an integer (1, 2, or 3) used to control the convergence of the integration. A value of 1 requires convergence to 10%, 2 requires convergence to 1%, and 3 requires convergence to 0.1%. In most cases the upset cross-section is not measured so precisely that the strictest convergence requirement is necessary to optimize the accuracy of the result. Execution is fastest using 1, 3 is pretty slow on a Mac or PC but is probably not too bad on a mainframe.

*/

/* The parameter INTRP is the number of interpolation steps used between listed values of the upset cross-section. If faster execution is desired this may be set to 1. If a finer interpolation of the listed cross-section is desired, it may be set higher. I use 8. The upper and lower limits of the upset rate are calculated using the obvious upper and lower bounds of the upset cross-section. They

Monday, March 16, 1992 10:23 AM

```

are certainly conservative limits. The mid value is from this
interpolation. */
#define INTRP 8

double Qc;          /* critical charge in PICOCOULOMBS */

#define NUMXST 50      /* number of entries in following table */
double xsinp[2][NUMXST]; /* cross-section table as provided */

#define NUMSPC 200     /* number of LET entries in table below */
double spectra[2][NUMSPC]; /* LET spectrum as provided */

double Accuracy = 0.001; /* End the integration when two successive
                           integrals differ by less than this
                           amount as a fraction of the value.
                           Note that the error in the result of the
                           integral likely will be larger than this */

/* The following constants are used to adjust units in the program */
double funits = 0.0000000864; /* convert from /s m2 to /day um2 86400*1e-12 */
double Amult = 1.0e8;          /* convert area in square cm to square microns */
double Tmult = 0.233;          /* convert thickness from microns to mg/cm2 (Si) */
double pCpMeV = 0.0445;        /* picoCoulombs per MeV for energy deposit in Si */
double qmult;                  /* to convert E- to Q-dep. MeV/mg/cm2 to pC/micron.
                               Defined in main as the product of Tmult * pCpMeV */

double xsect[NUMXST][2];      /* xsinp data converted to my units */
double spect[NUMSPC][3];      /* spectra data converted to my units */
double x, y, z, nbits, Xs, dQc, dXs;
double rnt[INTRP], ont[INTRP];
int nspect, ndevxs;

FILE *po;

main()
{
    int i, j, k, ntrp, index, model;
    double sqrt(), pldset(), myflux();
    double lmin, lmax, eff, lasteff, lastleff, lastueff;
    double umid, lmid, upper, lower, result;
    char kar, title[NUMSPC];

    qmult = Tmult * pCpMeV; /* define conversion from MeV/mg/cm2 to pC/um */

    po = fopen("SEU_CALC", "w");

    printf("\f SEU rate calculator - 1-D integration method");
    printf("\n Integral over experimental cross-section data.");
    printf("\n Version 1.0 D.L. Chenette 8 November 1991 \n");

    k = 0;
    do {
        printf("\n\n Enter convergence precision parameter (1=>low, 2=>medium, 3=>high):

```

Monday, March 16, 1992 10:23 AM

```

    scanf("%d",&i);
    k += 1;
    } while(k < 4 && (i<1 || i>3));
if (k >= 4) {
    printf("\n Input error. Got %d Should be 1 2 or 3");
    goto finish;
}
if (i == 1) Accuracy = 0.1;
if (i == 2) Accuracy = 0.01;
if (i == 3) Accuracy = 0.001;

if (po != NULL) {
    fprintf(po,"f  SEU rate calculator - 1-D integration method");
    fprintf(po,"n  Integral over experimental cross-section data.");
    fprintf(po,"n  Version 1.0  D.L. Chenette  8 November 1991 \n");
    fprintf(po,"n  Convergence parameter = %d. Integration consistency = %g\n",
        i,Accuracy);
}

if ((nspect = input_list("LET_SPECTRUM","LET",NUMSPC,spectra[0],spectra[1],title)) > 1
    printf("\n LET spectrum used in this run:");
    printf("\n %s",title);
    printf("\n  LET (MeV/micron)      Flux (/m2 s sr)");
if (po != NULL) {
    fprintf(po,"n LET spectrum used in this run:");
    fprintf(po,"n %s",title);
    fprintf(po,"n  LET (MeV/micron)      Flux (/m2 s sr)");
}

for(i=0;i<nspect;i++) { /* convert LET spectrum units */
    printf("\n  %12.4e      %12.4e",spectra[0][i],spectra[1][i]);
    if (po != NULL) {
        fprintf(po,"n  %12.4e      %12.4e",spectra[0][i],spectra[1][i]);
    }
    /* convert LET (MeV/micron) to charge dep (pC/micron) */
    spect[i][0] = spectra[0][i] * pCpMeV;
    spect[i][1] = spectra[1][i];
    if (i < nspect-1) spect[i][2] =
        (spect[i+1][1] - spect[i][1]) / (spect[i+1][0] - spect[i][0]);
    /* Note: conversion of flux to per sq micron and day
       is done only in printout at end to keep
       numbers larger (comparable to 1) */
    if ((i > 0) && (spectra[0][i-1] > spectra[0][i] ||
        spectra[1][i-1] < spectra[1][i])) {
        printf("\n Error in LET_SPECTRUM file. Must be monotonic");
        goto finish;
    }
}

printf("\n LET spectrum set.  %d values tabulated\n",nspect);
if (po != NULL) {
    fprintf(po,"n LET spectrum set.  %d values tabulated\n",nspect);
}
}

```

```

else {
    printf("\n LET Spectrum not found");
    goto finish;
}

if ((input_list("DEVICE", "X=", 0, &x, &x, title) == -1) &&
    (input_list("DEVICE", "Y=", 0, &y, &y, title) == -1) &&
    (input_list("DEVICE", "Z=", 0, &z, &z, title) == -1) &&
    (input_list("DEVICE", "N=", 0, &nbits, &nbits, title) == -1) &&
    ((ndevxs = input_list("DEVICE", "LET", NUMXST, xsinp[0], xsinp[1], title)) > 0) &&
    x > 0.0 && y > 0.0 && z > 0.0 && nbits > 0.0) {
    printf("\n Device Upset Cross-Section Tabulation");
    printf("\n %s", title);
    printf("\n   LET (MeV/mg/cm2)      Cross-section (cm2/device)");
    if (po != NULL) {
        fprintf(po, "\n Device Upset Cross-Section Tabulation");
        fprintf(po, "\n %s", title);
        fprintf(po, "\n   LET (MeV/mg/cm2)      Cross-section (cm2/device)");
    }
    for(i=0; i<ndevxs; i++) { /* convert cross-section data */
        printf("\n   %12.4e      %12.4e", xsinp[0][i], xsinp[1][i]);
        if (po != NULL) {
            fprintf(po, "\n   %12.4e      %12.4e", xsinp[0][i], xsinp[1][i]);
        }
        /* convert LET (MeV/mg/cm2) to charge deposit in pC */
        xsect[i][0] = xsinp[0][i] * qmult * z;
        /* cross-section/device to upset efficiency per bit */
        /* by dividing by number of bits and area per bit */
        xsect[i][1] = xsinp[1][i] / (x * y * nbits / Amult);
        /* calculate slopes for linear interpolations later */
        if ((i > 0) && (xsinp[0][i-1] > xsinp[0][i] ||
            xsinp[1][i-1] > xsinp[1][i])) {
            printf("\n Error in DEVICE data file. Must be monotonic");
            goto finish;
        }
    }
    printf("\n Sensitive region (microns): X = %.2f Y = %.2f Z = %.2f", x, y, z);
    printf("\n Number of bits per device = %.0f", nbits);
    printf("\n Cross section set.  %d values tabulated\n\n", ndevxs);
    if (po != NULL) {
        fprintf(po, "\n Sensitive region (microns): X = %.2f Y = %.2f Z = %.2f", x, y, z);
        fprintf(po, "\n Number of bits per device = %.0f", nbits);
        fprintf(po, "\n Cross section set.  %d values tabulated\n\n", ndevxs);
    }
}

else {
    printf("\n Device description file not found or in error");
    printf("\n Sensitive region (microns): X = %.2f Y = %.2f Z = %.2f", x, y, z);
    printf("\n Number of bits per device = %.0f", nbits);
    printf("\n %d cross-section values found\n\n", ndevxs);
    goto finish;
}

```

Monday, March 16, 1992 10:23 AM

```

/* range of pathlength integral is 0 to maximum pathlength in region */
    lmin = 0.0;
    lmax = pldset(x,y,z);
/*
printf("\n Press return to continue");
scanf("%c",&kar);
*/
    lastleff = 0.0;
    lastueff = 0.0;
    lasteff = 0.0;
    lower = 0.0;
    upper = 0.0;
    umid = 0.0;
    lmid = 0.0;
    for(i=0;i<ndevxs;i++) {      /* do all parts of cross-section */
        Qc = xsect[i][0];
        Xs = xsect[i][1];
        if (i < ndevxs-1) {
            dQc = (xsect[i+1][0] - xsect[i][0])/((double) INTRP);
            dXs = (xsect[i+1][1] - xsect[i][1])/((double) INTRP);
        }
        printf(" Part%3d of%3d   Qc=%11.4e   Xs=%11.4e",
            i+1,ndevxs,Qc,Xs);
        k = ((dQc <= 0.0) || (i == ndevxs-1)) ? 1 : INTRP;
        for(j=0;j<k;j++) { /* do interpolation steps */
            model = do_int(Accuracy,lmin,lmax,rnt+j,ont+j);
            Qc += dQc;
            Xs += dXs;
        }
        printf(" I=%11.3e\n",rnt[0]);
        upper += rnt[0] * (xsect[i+1][1] - lastueff);
        lastueff = xsect[i+1][1];
        lower += rnt[0] * (xsect[i][1] - lastleff);
        lastleff = xsect[i][1];
        lmid += rnt[0] * (xsect[i][1] - lasteff);
        lasteff = xsect[i][1] + dXs * (k - 1);
        if (k > 1) {
            for(j=0;j<k;j++) umid += rnt[j] * dXs;
            for(j=1;j<k;j++) lmid += rnt[j] * dXs;
        }
    }
    printf("\n L=%11.4e U=%12.4e lmid=%12.4e umid=%12.4e",
        lower, upper, lmid, umid);
    lower *= funits;
    upper *= funits;
    umid = (umid + lmid) * funits / 2;
    printf("\n\n Integration complete.");
    printf("\n Result =%12.4e upsets per bit per day",umid);
    printf("\n Extreme limit bounds are%12.4e to%12.4e",lower,upper);
    printf("\n Result =%12.4e upsets per device per day",umid*nbits);
    printf("\n Extreme limit bounds are%12.4e to%12.4e",
        lower*nbits,upper*nbits);
    if (po != NULL) {

```

Monday, March 16, 1992 10:23 AM

```

        fprintf(po, "\n\n Integration complete.");
        fprintf(po, "\n Result =%12.4e upsets per bit per day", umid);
        fprintf(po, "\n Extreme limit bounds are%12.4e to%12.4e", lower, upper);
        fprintf(po, "\n Result =%12.4e upsets per device per day", umid*nbits);
        fprintf(po, "\n Extreme limit bounds are%12.4e to%12.4e",
                lower*nbits, upper*nbits);
    }

finish:
printf("\n End of run. \n\n");
if (po != NULL) fclose(po);
}

double integrand(l) /* this is the function called by twodint() */
double l;
{
    double pldt(), myflux();
    if (l <= 0.0) return(0.0);
    return(pldt(l) * myflux(Qc/l));
}

double myflux(L) /* Calculate flux at LET = L */
double L; /* L in units of pC/micron */
{
    int i;
    static int oldi = 0;

    if (L >= spect[oldi][0] && L <= spect[oldi+1][0]) {
        i = oldi;
    }
    else {
        if (L <= 0.0 || L >= spect[nspect-1][0]) return(0.0);
        for(i=nspect-2; i>0; i--) if (L > spect[i][0]) break;
    }
    oldi = i;

    return(spect[i][1] + spect[i][2] * (L - spect[i][0])); /* calc flux at L */
}

double Igauss(a,b) /* do Gaussian integral of integrand from a to b */
double a,b;
{
    int j;
    double xr, xm, dx, s;
    double integrand();
    static double x[]={0.0, 0.1488743389, 0.4333953941,
                        0.6794095682, 0.8650633666, 0.97390652};
    static double w[]={0.0, 0.2955242247, 0.2692667193,
                        0.2190863625, 0.1494513491, 0.06667134};

    xm=0.5*(b+a);
    xr=0.5*(b-a);
    s=0;

```

```
    for (j=1;j<=5;j++) {
        dx=xr*x[j];
        s += w[j]*(integrand(xm+dx)+integrand(xm-dx));
    }
    return(s *= xr);
}

int do_int(Acc,mn,mx,res,old)
double Acc, mn, mx, *res, *old;
{
    double dx, answer, panswer, xmin, xmax;
    double min, max, eps, ddx, trint, ntrint;
    double rpart, opart, integrand(), lgauss();
    int i, section, ntr;

    ntr = 20;
    ntrint = (double) ntr;
/*
    printf("\n Integrate from %12.4e to %12.4e\n",mn,mx);
    if (po != NULL) {
        fprintf(po,"\n Integrate from %12.4e to %12.4e\n",mn,mx);
    }
*/
    dx = (mx - mn) / ntrint;
    for(i=0;i<=ntr;i++) {
        xmin = xmax;
        xmax = mn + dx * ((double) i);
        ddx = eps;
        eps = integrand(xmax);
        if (i == 0) {
            trint = 0.0;
        }
        else {
            trint += dx * (ddx + eps) / 2.0;
        }
/*
        printf(" x =%12.4e   f =%12.4e\n",xmax,eps);
        if (po != NULL) {
            fprintf(po," x =%12.4e   f =%12.4e\n",xmax,eps);
        }
*/
    }
/*
    printf(" %d point trapedoid integral =%12.4e\n",ntr,trint);
    if (po != NULL) {
        fprintf(po," %d point trapedoid integral =%12.4e\n",ntr,trint);
    }
*/
    rpart = 0.0;
    opart = 0.0;
    answer = 0.0;
    ddx = (mx - mn) / ntrint;
    for(section=0;section<ntr;section++) {
```


Monday, March 16, 1992 10:23 AM

```

min = mn + dx * ((double) section);
max = min + dx;
for (i=1; i<600; i+=2) {
    dx = (max - min) / ((double) i);
    panswer = answer;
    answer = 0.0;
    xmin = min;
    while (xmin < max) {
        xmax = xmin + dx;
        if (xmax > max) xmax = max;
        answer += Igauss(xmin, xmax);
        xmin = xmax;
    }
/*
    if ((i > 1 && answer != 0.0) || (i > 2)) {
*/
        if (answer != 0.0) eps = (answer - panswer) / answer;
        else eps = -1.0;
/*
        printf(" %5d %12.4e %12.4e %12.4e\n", i, panswer, answer, eps);
        if (po != NULL) {
            fprintf(po, " %5d %12.4e %12.4e %12.4e\n",
                i, panswer, answer, eps);
        }
*/
        if (fabs(eps) < Acc || fabs(eps*answer/trint) < Acc/10.0) break;
/*
    }
*/
    }
    rpart += answer;
    opart += panswer;
}
*old = opart;
*res = rpart;
return(i/2);
}

```

Monday, March 16, 1992 10:22 AM

/* INPUT_LIST.C

Version 1.0

Purpose: Read any listings

Coded by D.L. Chenette, 8 November 1991

Change log:

*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAXLEN 250

int input_list(ifname, key, maxn, col1, col2, title)

char *ifname, *key, *title;

int maxn;

double *col1, *col2;

{
char s[MAXLEN], *c, *d;

int k;

double *c1, *c2;

FILE *fp;

k = 0;

if (ifname && (fp = fopen(ifname, "r")) != NULL) {

/*

printf("\n Reading data from file: %s", ifname);

*/

if (*key) {

while(fgets(s, MAXLEN, fp)) {

c = s;

d = key;

while(*c && *c != *d) c++;

while(*d && *c && (*c == *d)) {

c += 1;

d += 1;

}

if (maxn == 0 && *d == '\0') {

while(*c && (*c < '-' || *c > '9')) c++;

if (*c) *col1 = atof(c);

return(-1);

}

if (*d == '\0') {

c = title;

d = s;

k = 80;

while((*c++ = *d++) && --k);

*c = '\0';

k = 0;

goto begin;

}

```
    }  
    printf("\n Input file %s did not contain key string %s",ifname,key);  
    goto done;  
}  
  
begin:  
    c1 = col1;  
    c2 = col2;  
    while(fgets(s,MAXLEN,fp)) {  
        c = s;  
        while(*c && (*c < '-' || *c > '9')) c++;  
        if (*c) {  
            *c1 = atof(c);  
            while (*c && (*c >= '-' || *c == '+')) c++;  
            while(*c && (*c < '-' || *c > '9')) c++;  
            if (*c) {  
                *c2 = atof(c);  
                c1 += 1;  
                c2 += 1;  
                if (++k == maxn) goto done;  
            }  
        }  
    }  
}  
  
done:  
    if (fp == NULL)  
        printf("\n Input file named %s cannot be opened for reading",ifname);  
    else fclose(fp);  
  
    return(k);  
}
```

Monday, March 16, 1992 10:32 AM

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double dist[1002]; /* pathlength distribution calculator method taken from */
double smax;       /* Pickel & Blandford, IEEE Trans.Nuc.Sci.NS-27, Apr.1980 */
                /* for right rectangular prism geometry */
                /* coded by D. L. Chenette 25 June 1987 */
double pldset(zz1, zz2, zz3) /* set up pathlength distribution table */
double zz1, zz2, zz3;       /* PLD is tabulated every 0.1% of maximum */
{
    /* pathlength (1001 points including max) */
    int i;
    /* returns maximum pathlength in volume */
    double pld(), sqrt();

    smax = sqrt(zz1 * zz1 + zz2 * zz2 + zz3 * zz3);
    for(i=0; i<1001; i++) {
        dist[i] = pld((double) smax*i/1000.0, zz1, zz2, zz3);
    }
    dist[1001] = 0.0;
    printf("\n Pathlength distribution set for right rectangular prism.");
    printf("\n Dimensions of sensitive region: x = %g y = %g z = %g",
        zz1, zz2, zz3);
    printf("\n Maximum pathlength through region = %g \n\n", smax);
    return(smax);
}

double pldt(s) /* return tabulated value of pathlength distribution */
double s;      /* for pathlength s through right rectangular prism */
{
    /* must have called pldset(x,y,z) first to setup table */
    int i;
    double x;
    if (s < 0.0 || s > smax) return(0.0);
    x = 1000.0 * s / smax;
    i = (int) x;
    return(dist[i] + (dist[i+1] - dist[i]) * (x - i));
}

double pld(s, zz1, zz2, zz3) /* return pathlength distribution for rect solid */
double s, zz1, zz2, zz3;     /* s = pathlength, zz1, zz2, zz3 are dimensions */
{
    /* pld is normalized s.t. integral over all s is */
    int i;
    /* equal to G-factor = 4*pi*(average proj. area) */
    double atan(), acos(), sqrt(), pow(); /* G-f = 2*pi*(zz1*zz2+zz1*zz3+zz2*zz3) */
    double x, y, z, r, r2, g, qsz, fs;
    double x2, y2, z2, xy2, xz2, xy, xz, s2, s3, ssz, ssz2;

    fs = 0.0;
    if (zz1 <= 0.0 || zz2 <= 0.0 || zz3 <= 0.0 || s < 0.0) return(fs);
    if (s > sqrt(zz1*zz1 + zz2*zz2 + zz3*zz3)) return(fs);

    s2 = s * s;
    s3 = s * s2;
    for(i=0; i<6; i++) {
        if (i == 0) {

```

```

    x = zz1;
    y = zz2;
    z = zz3;
}
else if (i == 1) {
    x = zz1;
    y = zz3;
    z = zz2;
}
else if (i == 2) {
    x = zz2;
    y = zz1;
    z = zz3;
}
else if (i == 3) {
    x = zz2;
    y = zz3;
    z = zz1;
}
else if (i == 4) {
    x = zz3;
    y = zz1;
    z = zz2;
}
else if (i == 5) {
    x = zz3;
    y = zz2;
    z = zz1;
}
x2 = x * x;
y2 = y * y;
z2 = z * z;
r2 = x2 + y2 + z2;
r = sqrt(r2);
xz2 = x2 + z2;
xz = sqrt(xz2);
xy2 = x2 + y2;
xy = sqrt(xy2);
if (s > z) {
    ssz2 = s2 - z2;
    ssz = sqrt(ssz2);
}
else {
    ssz2 = 0.0;
    ssz = 0.0;
}
if (s2 > xz2) qsz = sqrt(s2 - xz2);
else qsz = 0.0;
if (s < z) {
    g = y2/(xz2 * r) + 2.0 * (r - xz)/(xz * r) + r/xy2 - xz/x2
        + y2 * z/(x2 * xy2) - 1.125 * y2 * s/(r2 * xz2);
    g *= 2.0 * x2 / 3.0;
}

```

```

else if (s < xz) {
    g = y2/(xz2 * r) + 2.0 * (r - xz)/(xz * r) + r/xy2 - xz/x2
        - 1.125 * y2 * s/(r2 * xz2)
        - ssz * ssz2 * (x + (y2 - x2)/xy) / (s3 * x2)
        + 3. * z2 * y2 * (1. - z2/(3. * s2))/(2. * s * x2 * xy2)
        + 1.125 * ssz2 * ssz2 * y2 / (x2 * s3 * xy2);
    g *= 2.0 * x2 / 3.0;
    g += x * y * z2 * atan(y/x) / s3
        - z2 * ssz * ((y2 - x2)/xy + x)/s3 + z2*ssz2*y2/(2.*s3*xy2);
}
else if (s < r) {
    g = y2/(xz2 * r) - y * qsz/(s * xz2) - 2./r + r/xy2
        - s/ssz2 + 2./s - 1.125*s*(xy2/r2 - ssz2/s2)/z2
        - ssz * ssz2 * ((y2-x2)/xy - (y*qsz - x2)/ssz)/(s3*x2)
        + (3./(2.*x2))*(0.75*s - z2/(2.*s) + 5.*z2*z2/(12.*s3))
        * (y2/xy2 - (s2 - xz2)/ssz2);
    g *= 2.0 * x2 / 3.0;
    g += (x*y*z2/(s*s2))*(atan(y/x) - acos(x/ssz))
        - (z2/s3)*ssz*((y2-x2)/xy - (y*qsz - x2)/ssz)
        + (z2/(2.*s3)) * ssz2 * (y2/xy2 - (s2-xz2)/ssz2);
}
else g = 0.0;
fs += g;
}
return(8.0 * fs);
}

```