CALIBRATION OF THE DRAGON DSSSD END DETECTOR

HEATHER CRAWFORD
SIMON FRASER UNIVERSITY

The Double Sided Silicon Strip Detector (DSSSD)

The double sided silicon strip detector, or DSSSD, a position sensitive, segmented semi-conductor diode detector, is one of the options commonly used as an end detector at the DRAGON facility[1]. This detector, often used in conjunction with another detector, such as an MCP (micro-channel plate) can offer a wealth of information during experiments, including the number and energy of particles detected, as well as positional information, and when used in tandem with another detector, local timing information.

The DSSSD consists of 16 front strips orthogonal to 16 back strips, which creates a pixilated detection surface, allowing position information to be extracted for each particle incident on the detector. However, since each strip uses its own electronics, an important step before the detector can be used in an experiment is calibration to ensure all channels (adc and tdc channels) are gain and offset matched, as well as energy calibrated.

Calibration Experimental Set-Up

The DSSSD is typically calibrated using a triple alpha (^{241}Am / ^{239}Pu / ^{244}Cm) source placed just inside the beam line upstream of where the DSSSD detector is located. This source emits alpha particles at three distinct energies between 5 and 6 MeV. Table 1 summarizes the alpha energies emitted from the triple alpha calibration source.

Isotope	**Emitted Alpha Energy**
^{239}Pu	5.1566 MeV
^{241}Am	5.4856 MeV
^{244}Cm	5.8048 MeV

TABLE 1: *Alpha energies from triple alpha calibration source.*

To calibrate the detector, the source was put in place within the beam line, and the detector was pumped down to vacuum. Under vacuum, a calibration run of approximately 60 minutes was taken. This run resulted in a DSSSD energy spectrum as is shown in figure 1. In this figure, the y-axis represents the strips of the detector, with the front strips numbered 1-16, while the back strips correspond to strips 17-32. Figure 2 represents a projection of this figure onto the x-axis, giving a clear picture of the lack of calibration.

[1] For a complete description of the DSSSD end detector, refer to C. Wrede's Masters Thesis (Simon Fraser University, 2003), available at http://dragon.triumf.ca/docs/Wredethesis.pdf.

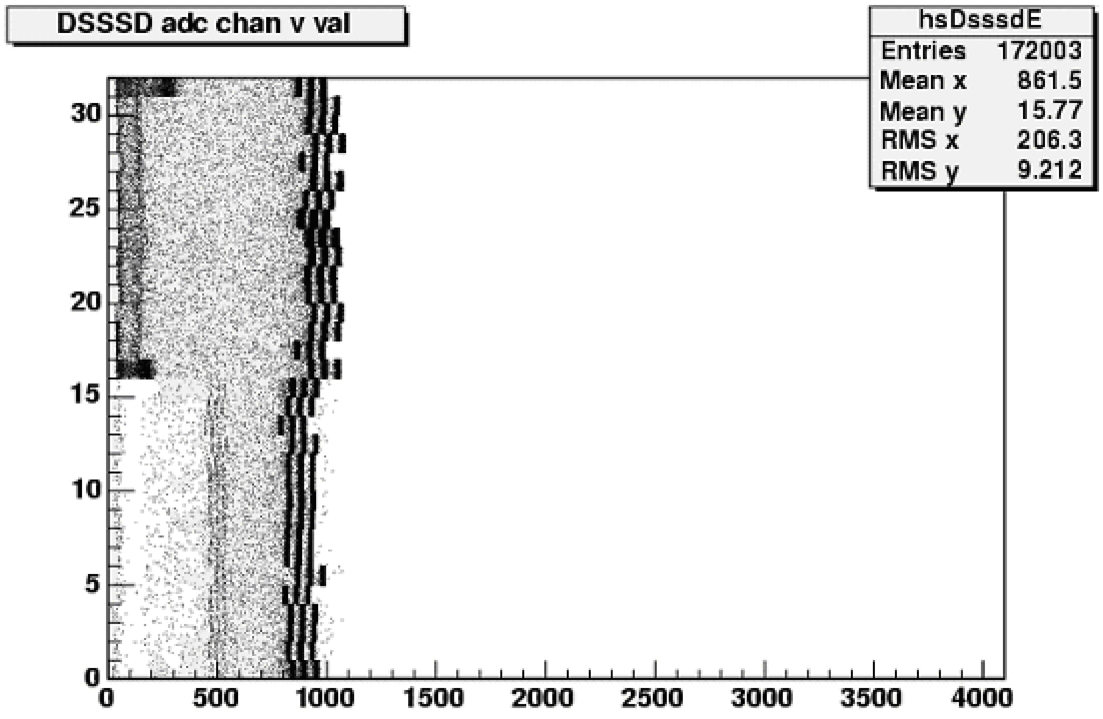

FIGURE 1: *Uncalibrated DSSSD adc energy spectrum*
(y-axis corresponds to DSSSD channel, x-axis to arbitrary energy units)

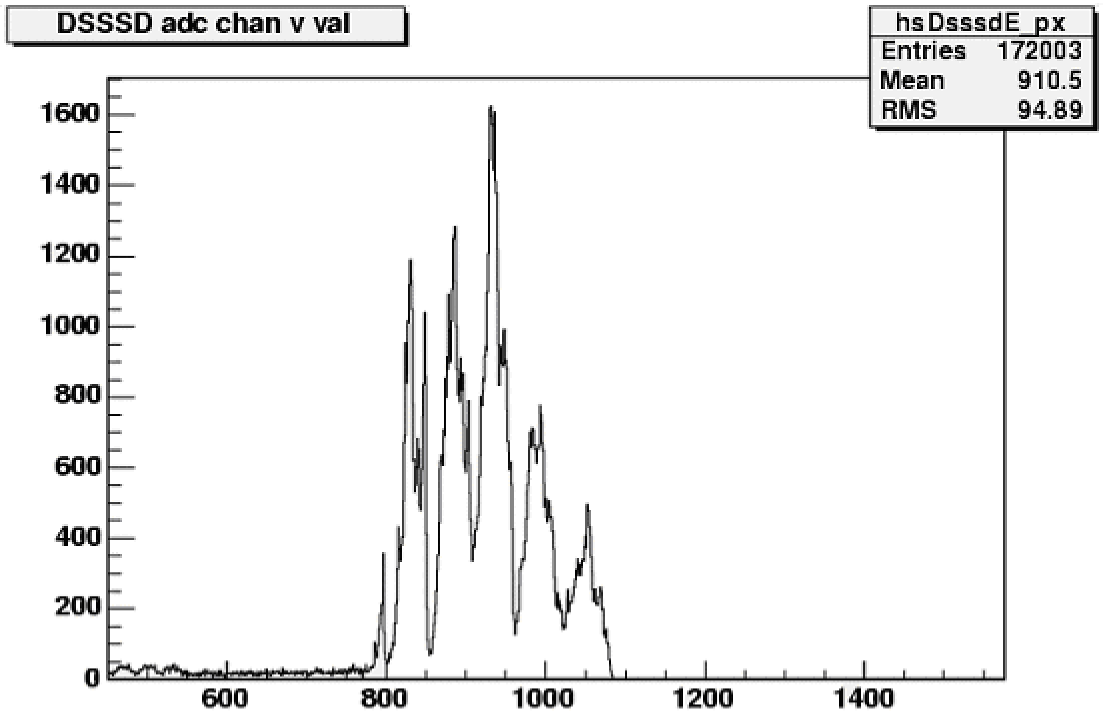

FIGURE 2: *Uncalibrated DSSSD adc energy spectrum projection of all strips onto x-axis*
(y-axis corresponds to number of counts, x-axis to arbitrary energy units)

Calibration Data Analysis

To process the data and actually obtain the calibration gain and offset values, each adc channel of the detector had to be individually evaluated and the three alpha peaks present in each spectra fit with Gaussians to determine their centroids and widths (sigma). Each tdc channel was also evaluated, and the single peaks in these spectra were fit with gaussians to determine their centroids and widths.

Once the peak locations for each channel had been determined, they were entered into a data file in a format as shown in appendix A, where peaks 1-3 represent the adc peaks in ascending energy order, and peak 4 represents the tdc peak for each DSSSD channel. A macro in ROOT (code for this macro can be found in appendix B) was then used to read in the raw data from a file, evaluate the energies of each alpha peak, taking into account energy loss through the dead layer of the detector, and fit a first-order polynomial to a plot of alpha energy versus peak position (for the adcs), after compensating for any previous calibration already in place on the detector strips. From these plots, the gains and offsets for each detector channel were calculated as the following:

$$Gain = Slope$$

$$Offset = -\frac{Intercept}{Slope}$$

To produce a calibrated DSSSD in terms of actual energy, the adc gains and offsets calculated by the macro are given as absolute, rather than relative. The macro also calculates relative tdc offsets by setting the lowest offset (lowest peak position) to be zero, and adjusting the others to this value.

The macro then outputs the adc gain and offsets and the tdc offset values to a user-specified file in the .sql format as is required for online and offline implementation of the calibration. (A sample file showing the .sql format is included in appendix C.)

Figure 5 shows the adc energy spectrum of the DSSSD after calibration was completed, while figure 6 shows the projection of this figure onto the x-axis.

Suggestions

The calibration of the DSSSD could, with a few alterations and additions to the macro in ROOT, be nearly fully automated. By combining the macro with peak-picking code, and altering it to read in the previous calibration from the online calibration database, DSSSD calibration could be as simple as putting the source in place and taking a run, then simply executing the calibration macro on the ROOT file corresponding to the run.

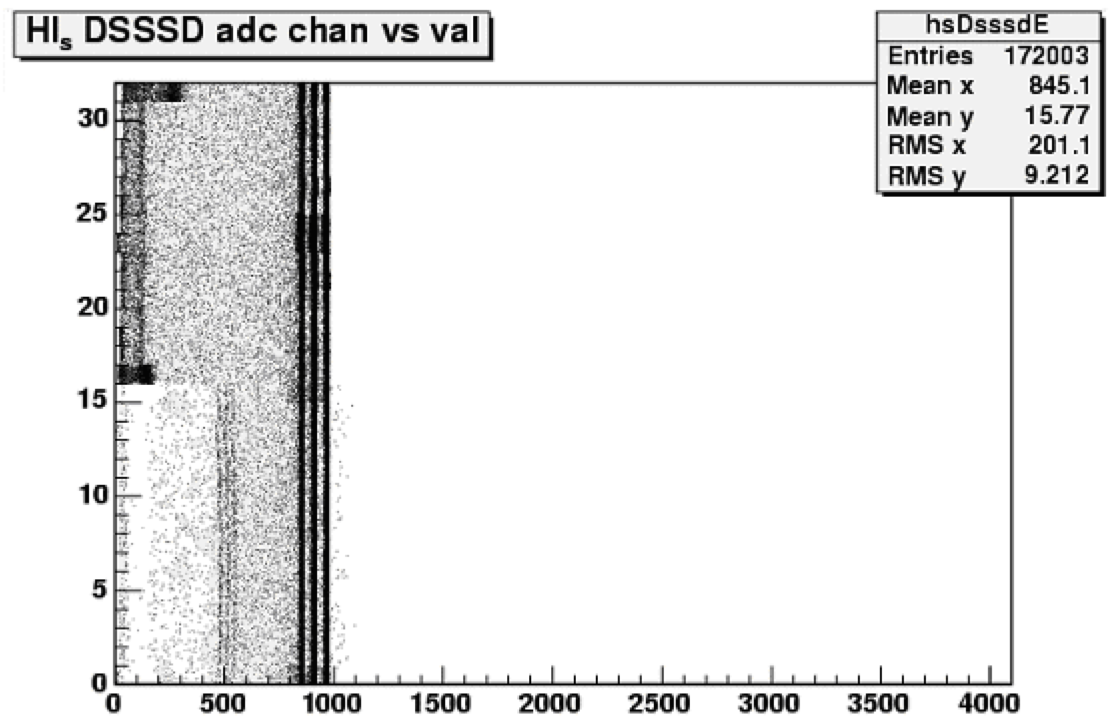

FIGURE 3: *Calibrated DSSSD adc energy spectrum*
(y-axis corresponds to DSSSD channel, x-axis to arbitrary energy units)

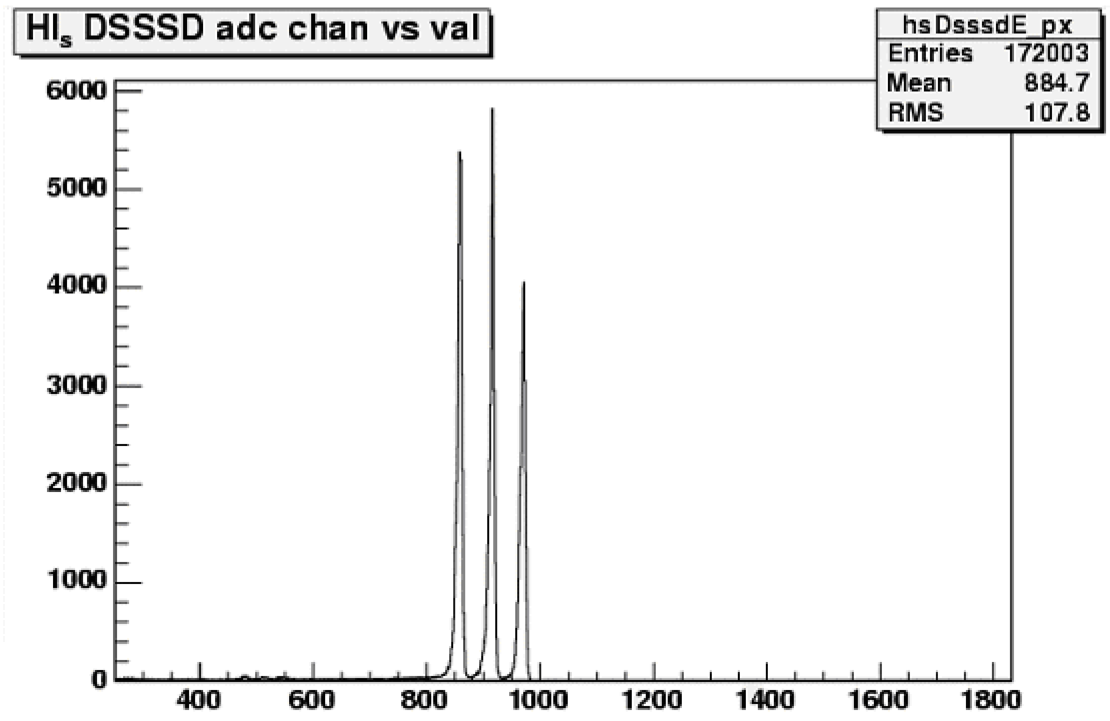

FIGURE 4: *Calibrated DSSSD adc energy spectrum projection of all strips onto x-axis*
(y-axis corresponds to number of counts, x-axis to arbitrary energy units)

APPENDIX A:
SAMPLE INPUT FILE TO DSSSD CALIBRATION ROOT MACRO

The input file to the macro written for the DSSSD calibration must be in the following format:

DSSSD Channel Peak Number Peak Centroid Peak Sigma

An example of this format is shown for the first 10 detector channels in the sample data file below. Peak numbers 1-3 correspond to the three alpha peaks in the adc spectrum in ascending energy order (or from left to right along the x-axis), while peak 4 refers to the tdc peak.

SAMPLE DATA INPUT FILE:

```
0 1 849.275 0.787351
0 2 904.136 2.44678
0 3 957.909 1.85708
0 4 92.5664 0.932194
1 1 830.939 2.43326
1 2 885.581 2.49435
1 3 938.308 2.26290
1 4 103.157 1.51877
2 1 839.360 2.40366
2 2 894.227 2.32094
2 3 947.272 1.98079
2 4 100.800 0.988619
3 1 840.842 2.25212
3 2 895.775 2.30407
3 3 950.324 1.97692
3 4 99.5750 1.18182
4 1 815.892 2.24592
4 2 868.263 2.17622
4 3 918.966 2.02146
4 4 98.5739 1.02519
5 1 873.807 2.61943
5 2 931.760 2.04580
5 3 986.212 2.08490
5 4 99.4288 0.927560
6 1 822.902 2.39097
6 2 877.475 2.28554
6 3 929.723 2.11610
6 4 99.8482 0.769236
7 1 825.407 1.76010
7 2 879.021 2.05863
7 3 930.344 2.07869
7 4 101.928 1.02649
8 1 830.312 2.35434
8 2 885.277 2.13440
8 3 937.700 2.32242
8 4 99.7052 1.32083
9 1 835.082 2.33670
9 2 890.255 2.19987
9 3 942.997 2.26356
9 4 97.2583 1.17234
```

Appendix B: C++ ROOT Macro For DSSSD Calibration Data

```
//
// Macro to read in raw DSSSD ADC and TDC triple alpha calibration data,
// correct for the calibration already in place and output a .sql file
// containing all of the calculated values.  To calibrate with the x
// axis representing energy, absolute values instead of relative values
// are used for ADC offset and gain.
//
// August 6, 2005 -- HCrawford
//

#include <iomanip>
#include <string>

//
// void dsssdcalib_sql(const char *filename = 0)
//

void dsssdcalib_sql(const char *filename = 0){
  const char *fName = filename;
  Float_t w, x, y, z;
  int precount = 0;
  ifstream in;

  //
  // Get data from file.
  //

  cout << "Given data file " << fName << endl;
  in.open(fName);

  if (in.is_open()) {
    while (! in.eof()) {
      in >> w >> x >> y >> z;

      //
      // Last read from file sets badbit, and w, x, y, z stay unchanged
      // making it look like we read the last line twice - the following
      // gets around it.
      //

      if (in.good()) {
       precount++;
       cout << "DSSSD Channel = " << w << ",\tPeak Number = " << x
            << ",\tx = " << y << ",\tex = " << z << endl;
      }
    }
    cout << "Found " << precount << " lines in data file" << endl;

    //
    // Reset ifstream.
    //

    in.clear(ios::goodbit);
    in.seekg(0,ios::beg);
    if (precount>0) {
      Int_t n = precount;
      Int_t nlines = 0;

      //
      // Create dynamic arrays to hold data points.
      //
      Float_t *dsssdChannel = new Float_t[n];
      Float_t *peakNumber = new Float_t[n];
      Float_t *peakChannel = new Float_t[n];
      Float_t *ex = new Float_t[n];
      Float_t *ey = new Float_t[n];
      Float_t *energy = new Float_t[n];
      Float_t *aEnergy = new Float_t[n];
      Float_t *slope = new Float_t[n];
      Float_t *intercept = new Float_t[n];
      Float_t *adcOffset = new Float_t[n];

      while (! in.eof()) {
       in >> w >> x >> y >> z;
       if (in.good()) {
         dsssdChannel[nlines] = w;
         peakNumber[nlines] = x;
         peakChannel[nlines] = y;
         ex[nlines] = z;
         ey[nlines] = 0;
         nlines++;
       }
      }
    }
    in.close();
  } else {

    //
    // Print error and return if we can't get the data.
    //

    cerr << "Could not open file " << fName << endl;
    cout << "Format of data file should be:" << endl
        << "DSSSD Channel Number, Peak Number, Channel Number (x),
 Error in Channel Number(ex)" << endl;
    return;
  }

  //
  // Now, we assign the appropriate alpha energies (adjusted for dead layer)
  // to peaks 1-3 (ADC peaks).
  //

  Float_t aPeaks[3] = {5.15659,5.48556,5.80477}; /* Alpha energies in MeV. */
  Float_t dLayer = 0.0005; /* Deadlayer thickness in mm. */
  Float_t aEloss[3] = {138.080,133.060,128.546}; /* Stopping Powers in
                                          MeV/mm. */
  for (int i=0; i<3; i++) {
    aEnergy[i] = aPeaks[i] - aEloss[i]*dLayer;
  }

  for(int j=0; j<=nlines; j++) {
    if (peakNumber[j] == 1) {
      energy[j] = aEnergy[0];
    } else if (peakNumber[j] == 2) {
      energy[j] = aEnergy[1];
    } else if (peakNumber[j]==3) {
      energy[j]= aEnergy[2];
    }
  }

  //
  // Now, we want to correct for the calibration previously applied to the
  // front strips and back strips if applicable.
  //

  Float_t AdcOriginalGain[32] =
{0.0058289,0.0056741,0.0057017,0.0057920,0.0060774,0.0056703,0.0057680,0.0057527,0.0
```

```
058462,0.0059524,0.0057094,0.0057760,0.0058997,0.0059502,0.0057586,0.0058510,0.00548
69,0.0058546,0.0054409,0.0053405,0.0055712,0.0055670,0.0054087,0.0055959,0.0057272,0
.0056191,0.0053962,0.0057526,0.0053185,0.0054459,0.0054098,0.0057271};
  Float_t AdcOriginalOffset[32] = {17,-2,4,4,10,6,-4,15,1,10,7,5,21,25,-5,1,4,-3,-
4,-10,4,-1,-7,20,0,1,-3,3,-7,-15,-13,-17};
  Float_t TdcOriginalOffset[32] =
{2,12,10,9,8,9,9,11,9,6,6,2,6,5,6,9,5,13,4,0,9,9,10,7,17,8,10,14,7,15,14,3};

  int j=0; /* Define counting variable for use in loop. */

  for(int i=0; i<128;) {
    peakChannel[i]=(peakChannel[i]/AdcOriginalGain[j])+AdcOriginalOffset[j];
    peakChannel[i+1]=(peakChannel[i+1]/AdcOriginalGain[j])+AdcOriginalOffset[j];
    peakChannel[i+2]=(peakChannel[i+2]/AdcOriginalGain[j])+AdcOriginalOffset[j];
    peakChannel[i+3]=(peakChannel[i+3])+TdcOriginalOffset[j];
    j++;
    i+=4;
  }

  //
  // Now open a file to write to and determine the needed parameters.
  //

  string output;

  cout << "Please enter a filename for the output calibration values. (i.e.
dsssd.sql)" << endl;
  cout << "The file format generated will be a .sql file. : ";
  cin >> output;

  ofstream calib;
  calib.open(output.c_str());

  //
  // Fit linear polynomial to ADC data & get ADC parameters (absolute, not
  // relative values).
  //

  int k=0; /* Define counting variable for loop. */
  for(int i = 0; i<nlines;) {
    cout << "\nCalculating fit for Channel #" << k << "\n" << endl;
    TGraphErrors *gr = new TGraphErrors(3, peakChannel+i, energy+i, ex+i,
                                ey+i);
    TF1 *fit = new TF1("fit", "pol1", 0, peakChannel[nlines]);
    gr->Fit("fit", "F");
    intercept[k] = fit->GetParameter(0);
    slope[k] = fit->GetParameter(1);
    adcOffset[k] = -intercept[k]/slope[k];
    i=i+4;
    k++;
  }

  //
  // Determine TDC relative offsets as well.
  //

  Float_t tdcLowOffset = peakChannel[3];

  for(int j=0; j<nlines; j++) {
    if(peakNumber[j]==4) {
      if(peakChannel[j]<tdcLowOffset) {
       tdcLowOffset = peakChannel[j];
      }
    }
  }

  int d=0; /* Another counting variable for loop. */
  Float_t *tdcOffset = new Float_t[32];

  for(int i=3; i<nlines;) {
    cout << "Calculating TDC Offset for Channel # " << dsssdChannel[i]
        << "..." << endl;
    tdcOffset[d] = peakChannel[i] - tdcLowOffset;
    i=i+4;
    d++;
  }

  //
  // Write the calibration parameters to file previously opened (round off
  // offset values to nearest integer). Enter in a .sql file format.
  //

  calib << "-- This is for the DSSSD detector," << endl;
  calib << "-- to use this script do, for example, the following:" << endl;
  calib << "-- % mysql -u dragon -p < dsssd.sql" << endl;
  calib << "USE dragon;\n" << endl;
  calib << "DROP TABLE IF EXISTS `DSSSD0`;" << "\n" << endl;
  calib << "CREATE TABLE `DSSSD0` (" << endl;
  calib << "  `detector channel`" << setw(19) << "int NOT NULL," << endl;
  calib << "  `adc map`" << setw(23) << "integer," << endl;
  calib << "  `adc offset`" << setw(20) << "integer," << endl;
  calib << "  `adc gain`" << setw(21) << "double," << endl;
  calib << "  `adc histogram channel`" << setw(9) << "integer," << endl;
  calib << "  `tdc map`" << setw(23) << "integer," << endl;
  calib << "  `tdc offset`" << setw(20) << "integer," << endl;
  calib << "  `tdc gain`" << setw(21) << "double," << endl;
  calib << "  `tdc histogram channel`" << setw(9) << "integer," << endl;
  calib << "  PRIMARY KEY (`detector channel`)" << endl;
  calib << ");\n" << endl;

  int g = 8;
  int h = 47;

  for (int k = 0; k < 32; k++) {
    calib << "INSERT INTO DSSSD0 VALUES (" << setw(2) << k << ", " << setw(2);
    if (k < 16){
      calib << k << ", " << setw(3) << TMath::Nint(adcOffset[k]) << ", "
           << setw(10) << slope[k] << ", " << setw(2) << k << ", " << setw(2)
           << k << ", " << setw(2) << TMath::Nint(tdcOffset[k]) << ", "
           << setw(2) << "1, " << setw(2) << k << ");" << endl;
    }else if (k >15 && k < 24){
      calib << g << ", " << setw(3) << TMath::Nint(adcOffset[k]) << ", "
           << setw(10) << slope[k] << ", " << setw(2) << k << ", " << setw(2)
           << g << ", " << setw(2) << TMath::Nint(tdcOffset[k]) << ", "
           << setw(2) << "1, " << setw(2) << k << ");" << endl;
    }else if (k > 23){
      calib << h << ", " << setw(3) << TMath::Nint(adcOffset[k]) << ", "
           << setw(10) << slope[k] << ", " << setw(2) << k << ", " << setw(2)
           << h << ", " << setw(2) << TMath::Nint(tdcOffset[k]) << ", "
           << setw(2) << "1, " << setw(2) << k << ");" << endl;
    }
    g++;
    h--;
  }

  cout << "\nData fitting complete. Fit parameters written to " << output << ".\n"
<< endl;
}
```

APPENDIX C:
SAMPLE .SQL FILE (FORMAT OF OUTPUT FOR CALIBRATION RESULTS)

```
-- This is for the DSSSD detector,
-- to use this script do, for example, the following:
-- % mysql -u dragon -p < dsssd.sql
USE dragon;

DROP TABLE IF EXISTS `DSSSD0`;

CREATE TABLE `DSSSD0` (
  `detector channel`      int NOT NULL,
  `adc map`               integer,
  `adc offset`            integer,
  `adc gain`              double,
  `adc histogram channel` integer,
  `tdc map`               integer,
  `tdc offset`            integer,
  `tdc gain`              double,
  `tdc histogram channel` integer,
  PRIMARY KEY (`detector channel`)
);

INSERT INTO DSSSD0 VALUES ( 0,  0,   6, 0.00578782,  0,  0,  3, 1,  0);
INSERT INTO DSSSD0 VALUES ( 1,  1,  -6, 0.00567031,  1,  1,  9, 1,  1);
INSERT INTO DSSSD0 VALUES ( 2,  2,   7, 0.00574714,  2,  2, 10, 1,  2);
INSERT INTO DSSSD0 VALUES ( 3,  3,  -2, 0.00577589,  3,  3,  8, 1,  3);
INSERT INTO DSSSD0 VALUES ( 4,  4,  11, 0.00609764,  4,  4,  7, 1,  4);
INSERT INTO DSSSD0 VALUES ( 5,  5,   8, 0.00569193,  5,  5,  8, 1,  5);
INSERT INTO DSSSD0 VALUES ( 6,  6,   1, 0.00581128,  6,  6,  9, 1,  6);
INSERT INTO DSSSD0 VALUES ( 7,  7,  19, 0.00579301,  7,  7, 11, 1,  7);
INSERT INTO DSSSD0 VALUES ( 8,  8,   4, 0.00587246,  8,  8,  8, 1,  8);
INSERT INTO DSSSD0 VALUES ( 9,  9,   3, 0.00593479,  9,  9,  6, 1,  9);
INSERT INTO DSSSD0 VALUES (10, 10,   1, 0.00569254, 10, 10,  5, 1, 10);
INSERT INTO DSSSD0 VALUES (11, 11,   3, 0.00577536, 11, 11,  2, 1, 11);
INSERT INTO DSSSD0 VALUES (12, 12,  10, 0.00585481, 12, 12,  6, 1, 12);
INSERT INTO DSSSD0 VALUES (13, 13,   6, 0.00583062, 13, 13,  5, 1, 13);
INSERT INTO DSSSD0 VALUES (14, 14,  -3, 0.00579775, 14, 14,  6, 1, 14);
INSERT INTO DSSSD0 VALUES (15, 15,   0, 0.00586826, 15, 15,  8, 1, 15);
INSERT INTO DSSSD0 VALUES (16, 24,  -4, 0.00540262, 16, 24,  4, 1, 16);
INSERT INTO DSSSD0 VALUES (17, 25,  -2, 0.00583915, 17, 25, 12, 1, 17);
INSERT INTO DSSSD0 VALUES (18, 26,   2, 0.00548208, 18, 26,  3, 1, 18);
INSERT INTO DSSSD0 VALUES (19, 27, -10, 0.00533219, 19, 27,  0, 1, 19);
INSERT INTO DSSSD0 VALUES (20, 28,  12,  0.0055949, 20, 28,  4, 1, 20);
INSERT INTO DSSSD0 VALUES (21, 29,  -4,  0.0055154, 21, 29,  6, 1, 21);
INSERT INTO DSSSD0 VALUES (22, 30,   0, 0.00542504, 22, 30,  6, 1, 22);
INSERT INTO DSSSD0 VALUES (23, 31,  11, 0.00552072, 23, 31,  7, 1, 23);
INSERT INTO DSSSD0 VALUES (24, 23,   1, 0.00570639, 24, 23, 15, 1, 24);
INSERT INTO DSSSD0 VALUES (25, 22,   6, 0.00564055, 25, 22,  8, 1, 25);
INSERT INTO DSSSD0 VALUES (26, 21, -10,  0.0053235, 26, 21, 10, 1, 26);
INSERT INTO DSSSD0 VALUES (27, 20,  14, 0.00578314, 27, 20, 14, 1, 27);
INSERT INTO DSSSD0 VALUES (28, 19,  -6, 0.00531289, 28, 19,  6, 1, 28);
INSERT INTO DSSSD0 VALUES (29, 18,  -7, 0.00546767, 29, 18, 14, 1, 29);
INSERT INTO DSSSD0 VALUES (30, 17, -11, 0.00541298, 30, 17, 12, 1, 30);
INSERT INTO DSSSD0 VALUES (31, 16,  -6, 0.00573813, 31, 16,  4, 1, 31);
```