LuckyDoll, an analysis package for the AIDA detector

Vi Ho Phong

A preliminary manual
May 17, 2017

Contents

1 Changelog

1.1 May 13, 2017

The correlation scaler signals are now included in the simplified output root file with ID=6.

1.2 February 9, 2017

LuckyDoll now can read the GZIP file directly. Use the new option "-gz 1" to read gz file! New Version requires an installed BOOST library $\geq$ 1.42.0

1.3 October 1, 2016

New event builder algorithm using sucessive ADC word is added. Use the excutable ./aidanew or ./aidafullnew

1.4 November 7, 2016

Add the FEE number and channel number to the full tree. Changed default (clear values) of the channel identification (ffee, fch, fxy, fz fid) to -1 to avoid potential problem.

1.5 November 5, 2016

Change the help menu in ./aidafull For ./aidafull and ./aida: fix bug that force users to enter threshold and calibration file even if they don't want to.

1.6 October 28, 2016

A faster clustering function has been added. About 25% improvement in the overall perfomance.

1.7 October 26, 2016

An almost final version released

2 Introduction

2.1 What This is About?

LuckyDoll is a data analysis package which is used for analyzing the data from AIDA detector.

The AIDA detector is a new generation implantation detector which is specificly designed for decay experiment at fast RI beam facilities. More information about AIDA can be found at [2]

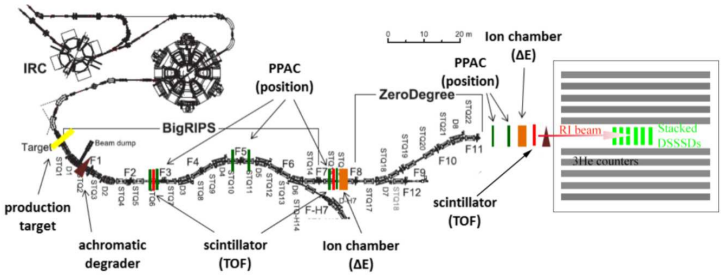

Figure 1: A schematic view of the BRIKEN setup

The developement of the LuckyDoll package is gorverned by a number of principles:

- Capatable to analyze a large amount of data from both implantation and decay events within a reasonable computing time so that it can be used for semi-online analysis during the experiment
- Comprehensive data structure
- Easy to understand, easy to use
- Produce output data with a mimized the number of the background and unwanted events.

2.2 LuckyDoll main classes and what are they good for

LuckyDoll package is divded into several classes, each is programed to perform a particluar task

AIDAUnpacker class handles the decoding of the MIDAS data and convert it into the "hits" seen by AIDA ADC

BuildAIDAEvent class is used to reconstruct the decay and implantation events from the energy deposition in the strips of the sillicon detector. The reconstruction is based on timing and spatial correlation. The implementation of the class is as follow:

- Open files, initialize the Event Builder object

```
TFile* ofile = new TFile(OutFile,"recreate");
 ofile->cd();
```

```
//! Book tree and histograms
TTree* treeion=new TTree("ion","tree ion");
TTree* treebeta=new TTree("beta","tree beta");
TTree* treepulser=new TTree("pulser","tree pulser");


BuildAIDAEvents* evts=new BuildAIDAEvents;
evts->SetVerbose(Verbose);
evts->SetFillData(true);
evts->BookTree(treeion,treebeta,treepulser);
evts->SetMappingFile(MappingFile);
evts->SetThresholdFile(ThresholdFile);
evts->SetCalibFile(CalibrationFile);
evts->SetAIDATransientTime(TransientTime);
evts->SetEventWindowION(WindowIon);
evts->SetEventWindowBETA(WindowBeta);
evts->Init(R10_0);
```

- Event loop (similiar with analoop)

```
while(evts->GetNextEvent()){
if(evts->IsBETA())
evts->GetAIDABeta()->GetCluster(0)->GetHitPositionX();
else
evts->GetAIDAIon()->GetCluster(0)->GetHitPositionX();
evts->GetAIDAIon()->GetCluster(0)->GetEnergy();
evts->GetAIDAIon()->GetCluster(0)->GetTimestamp();
}
```

- Write tree and close files

```
    treeion->Write();
    treebeta->Write();
    treepulser->Write();
    ofile->Close();
```

The sorted and reconstructed data is stored in an object named "AIDA", which can be found in the source code:

```
//! aida time stamp (ealiest timestamp within event)
    unsigned long long faidats;
    //! type of event: 0 beta  1 ion
    short ftype;
    //! total multiplicity
    unsigned short fmult;
    //! x multiplicity
    unsigned short fmultx[NumDSSD];
    //! y multiplicity
    unsigned short fmulty[NumDSSD];
    //! x sum all energy
```

```
double fsumx[NumDSSD];
//! y sum all energy
double fsumy[NumDSSD];
//! total clusters
unsigned short fnclusters;
//! total clusters
unsigned short fnclustersz[NumDSSD];
//! max hit position
unsigned short fmaxz;
//!Threshold table
Double_t fdssd_thr[NumDSSD][NumStrXY];
//!Calibration table
Double_t fdssd_cal[NumDSSD][NumStrXY][2];
//! vector with the hits
vector<AIDAHit*> fhits;
//! vector with the clusters
vector<AIDACluster*> fclusters;
```

This object contains STL vectors of **"hits"** and **"clusters"**. The **"hits"** information is generally defined in the class "AIDAHit" which contains the channel identification (i.e. id, fee number, channel number, strips number and dssd number), energy, ADC value and timestamp for each self-triggered hit read out by DAQ. They are described in the source code:

```
//! energy range : (0: high gain, 1: low gain)
short frange;
//! translate into channel ID number
short fid;
//! translate into the DSSDs coordinator
short fxy;
short fz;
//! FEE information
short ffee;
short fch;

//! the energy lab system
double fen;
//! the raw adc value
int fadc;
//! the timestamp
unsigned long long fts;
//! the fast timestamp
unsigned long long ffastts;

//! current hits
unsigned short fhitsadded;
```

Based on the information collected within an event window (or a sequence of DAQ read-out) as a vector of hits, the event is reconstructed by using the spatial correlation. For that purpose, the term **"cluster"** is introduced, which represents the position where the energy deposition of the radiation takes place

in the DSSD. It can be energy deposition of the heavy ion which triggers the high gain branch of the AIDA electronics or the energy deposition of the beta particles emitted from the decay of the implanted radiactive isotope. The details on how to reconstruct the event based on spatial correlation (to make "clusters" from "hits") can be found in the section **??**.

The **"cluster"** is defined by the class "AIDACluster" as follows:

```
//! translate into the DSSDs coordinator
TVector3 fpos;

//! the energy lab system
double fsumenx;
double fsumeny;

//! number of hit in one cluster
unsigned short fnx;
unsigned short fny;

//! the ealiest timestamp
unsigned long long fcts;

//! the ealiest fast timestamp
unsigned long long fcfastts;

//! current hits
unsigned short fclustersadded;
```

2.3 LuckyDoll main programs

The Lucky main programs are builted on top of the main classes to perform some paticular analysis jobs.

- **./aida(new)** : Unpack, calibrate, build aida events and produce a root file with minimum information which later can be used for the BRIKEN merger program.
- **./aidafull(new)** : Unpack, calibrate, build aida events and produce a root file with full information.
- **./aida2tree** : Unpack the raw data and produce a root file containing infomation about hits in FEE channels (uncalibrated)
- **./aida2histbkg** : Make histograms of ADC value versus strip number for the purpuse of determining the individual threshold for each AIDA strip.
- **./aida2histcalib** : Make histograms of ADC value versus strip number considering the pulser events (high multiplicity) or low energy(and low multiplicity) events for the purpuse of energy calibration using pulser or radioactive source.

3 LuckyDoll basic: How to run it and what is the data structure?

3.1 Installation

....

Prerequites

- **gcc**
- **cmake** version ≥ 3
- CERN **root** version ≥ 5.34
- BOOST version $\geq 1.42.0$

Installation Steps

```
git clone https://github.com/vihophong/LuckyDoll.git
cd LuckyDoll
cd ../
mkdir build_LuckyDoll
cd build_LuckyDoll
cmake ../LuckyDoll
make
```

3.2 Convert AIDA partial data

This main program "./aida" or "./aidanew" provides root file for IFIC merger program ®

- **./aida** uses the fix event builder window to reconstruct the beta, implantation and pulser events.
- **./aidanew** uses the flexible event builder window based on the read-out sequence of the AIDA DAQ to reconstruct the beta, implantation and pulser events. Users is recommened to use this program.

The help menu will be shown when there is no input argument specified by user:

```
$ ./aidanew
AIDA event builder
use ./aidanew with following flags:
        [-a     <char*    >: AIDA input list of files]
        [-o     <char*    >: output file]
        [-wd    <long long>: Fast Discriminator Scan window
        (default: 0 i.e no scan for fast discrimination)]
        [-v     <int      >: verbose level]
        [-map   <char*    >: mapping file]
        [-cal   <char*    >: calibration file]
        [-thr   <char*    >: threshold file]
        [-f     <int      >: fill data or not: 1 fill data 0 no fill
```

```
        (default: fill data)]
        [-ecut  <char*     >: specify energy cut file]
        [-ecorr <double    >: specify energy corrlation cut]
        [-gz    <int       >: input data from gz file: 1 enable 0 disable
        (default: disable)]
        [-rmode <int       >: Switch on(1) off(0) the position determination
        based on energy correlation ranking (default:on)]
        [-smult <int       >: DSSD multiplicity cut (default 10000)]
No AIDA input list of files given
```

The data structure is as follows:

```
typedef struct {
    unsigned long long T;         // Calibrated timestamp
    unsigned long long Tfast; // calibrated fast timestamp
    double E;     // Energy depostions = (EX+EY)/2
    double EX;// X strips energy deposition
    double EY; // Y strips energy deposition
    double x,y,z;// number of pixel for AIDA,
    int nx, ny, nz;// multiplicity of hits in x
    //and y strips, and dssd planes
    unsigned char ID;     // Detector type:
    //Aida ion = 4, AIDA beta = 5
} datatype;
```

which represents the **"cluster"** information in a simplified structure. For more information about how to extract the "cluster" information, refer to **Section 3.3**

Explanation of each input is as follows:

- **a : AIDA input list of files**: A text file contains list of the input raw data file to be processed. An example of this file is as follows:

  ```
  11
  aidaraw/R25_13
  aidaraw/R25_14
  aidaraw/R25_15
  aidaraw/R25_16
  aidaraw/R25_17
  aidaraw/R25_18
  aidaraw/R25_19
  aidaraw/R25_20
  aidaraw/R25_21
  aidaraw/R25_22
  aidaraw/R25_23
  ```

  Where the first line represents number of files to be processed. Followed lines indicate path to the input files (namely the sub runs).

  Tips: One can use shell-script to generate this file (see **Section 4**)

- **-o : output file**: Path to the output file.

- **-wd : Fast Discriminator Scan...**: If fast timming branch (using discriminator before shaping amplifier) in AIDA hardware is enabled, this option provides user-specified time window to look for this fast discriminator information.
- **-v: verbose level**: This option is still under developement.
- **-map: mapping file:**: A text file contains a "map" to convert the Front-End Electronics channel addess to physical DSSDs strip number. This file contains 5 columns:

  FEE No. FEE channel DSSD No. DSSD channel Channel eable/disable mask

  Note: The index number should start from 0. DSSD channel from 0 to 127 represents X strips and from 128 to 255 represents Y strips.
- **-cal: calibration file**: A text file tabulates the energy calibration parameters for each DSSD strip assuming linear polynomial function of energy versus channel number. This file contains 4 columns:

  DSSD No. DSSD channel Offset(keV) Gain(keV/channel)

  Note: If a common gain are assumed for all strips. It is straightforward to calculate the offset value (in keV) from the ADC offset value extracted from the pulser-walkthgrough run using equation: $Offset(keV) = -(ADCoffset) * Gain(keV/channel)$
- **-thr: threshold file**: A text file tabulates the individual channel (software) threshold cut by ADC channel for the low energy branch (i.e. All hits with ADC value less than the theshold will be discarded in the event builder). This file contains 3 columns:

  DSSD No. DSSD channel Threshold value (channel)
- **-f: fill data or not**: Option to enable/disable writing output data file.
- **-ecut: specify energy correlation cut**: Specify further threshold cut to be applied for events categoried as the low energy after event reconstruction. This file contains 3 columns:

  DSSD No. Energy threshold cut in X(keV) Energy threshold cut in Y(keV)
- **-ecorr: specify energy cut file**: Apply cut on energy correlation between Y and X strip for low energy (reconstructed) events: $Energy_Y/Energy_X < 1 + Ecorr$ and $Energy_Y/Energy_X > 1 - Ecorr$
- **-gz: input data from gz file**: Option to choose whether input raw data file are compressed (gz zipped) or decompressed.

- **-rmode: ranking mode enable/disable**: By default the reconstruction algorithm use so called "ranking model" to classify the real beta hit among "ghost hits" by sorting the energy correlation parameters. User can disable this feature, so that all combinations (including ghost hits) will be used.
- **-smult: DSSD multiplicity cut**: Set maximum allowed multiplicity for each DSSD.

Note:

- Examples of above configuration files can be found in the directory named "txt" in the source code.
- The event building window, transient time (for fix event builder window, "**./aida**" program) and fast discriminator scan window have been set by default (see above). Therefore, the input -wi -wb -wd -tt is not needed. if user don't want to change those value.
- If there is no calibration table specified by user the energy calibration will not be made, which is not good for the event correlation, especially the clustering algorithm.
- If there is no threshold table specified by user, the program will consider getting all information from AIDA (threshold = -10000) for all the channels.

3.3 Convert AIDA full data

The help menu will be shown when there is no input argument specified by user:

```
$ ./aidafullnew
AIDA event builder
use ./aidafullnew with following flags:
        [-a     <char*    >: AIDA input list of files]
        [-o     <char*    >: output file]
        [-wd    <long long>: Fast Discriminator Scan window
        (default: 0 i.e no scan for fast discrimination)]
        [-v     <int      >: verbose level]
        [-map   <char*    >: mapping file]
        [-cal   <char*    >: calibration file]
        [-thr   <char*    >: threshold file]
        [-f     <int      >: fill data or not: 1 fill data 0 no fill
        (default: fill data)]
        [-ecut  <char*    >: specify energy cut file]
        [-ecorr <double   >: specify energy corrlation cut]
        [-gz    <int      >: input data from gz file: 1 enable 0 disable
        (default: disable)]
        [-rmode <int      >: Switch on(1) off(0) the
        position determination based on energy
        correlation ranking (default:on)]
        [-smult <int      >: DSSD multiplicity cut (default 10000)]
No AIDA input list of files given
```

The resulting root file from **./aidafull** contains 3 trees: ion, beta and pulsers. The Branches and Leafs structure of those trees are identical. They are defined in the source code "AIDA.h" as a TObject named "AIDA". The "AIDA" object stores STL Vectors of the so called "hits" and "clusters" within a time window (or a sequence of DAQ read-out).

4 Using shell-script to generate file list automatically and run the program

The execution of the LuckyDoll main program can be simplified by using a simple shell script:

```
#!/bin/bash
if [ $# == 4 ]; then

    expr $3 - $2 + 1 > list/list_R$4_$1_$2to$3.txt
    for i in `seq $2 $3`;
    do
    ls -tr aidaraw/R$1_$i >> list/list_R$4_$1_$2to$3.txt
    done
    # run
    ./aidanew -a list/list_R$4_$1_$2to$3.txt -o rootfiles/
    aidatakechinew/$4_aida$1_$2to$3.root -map config/
    FEE_table_briken2016.txt -thr config/thr_r141_1cps.txt
    -cal config/cal_briken2016_r25_26.txt -wi 5000

else
    echo "usage: ./aidabuild_takechi.sh run_number
    subrun_number_low subrun_number_high yymmdd_hhmm"
fi
```

This script, which can be found at the directory named "txt" in the LuckyDoll source code, will generate a text file containing a list of data files to be processed in a sequence of AIDA sub runs (as an input of the LuckyDoll main program) and put it in the directory named "list", which should be the same directory as the excutable "aidanew" (one should create this directory before running the script). After that, the script will execute the main program to sort that AIDA raw data to the rootfile with the name:

```
$4_aida$1_$2to$3.root
Where $4, $1, $2 and $3 are following input arguments:
./aidabuild_takechi.sh run_number
subrun_number_low subrun_number_high yymmdd_hhmm
```

One should also specify the directory which store the raw data accordingly by changing the line:

```
ls -tr aidaraw/R$1_$i >> list/list_R$4_$1_$2to$3.txt
```

In this example we assumed the raw data directory to be "aidaraw" (one can,of course, make a soft link to a more complicated path).

References

[1] A. Tarifeño-Saldivia et al.: arXiv:1606.05544.

[2] http://www2.ph.ed.ac.uk/ td/AIDA/Information/information.html