SPS Digital Event Builder

Gordon McCann

1   Overview

This paper is aimed at giving a brief example of how to use the EventBuilder code. Note that this code is a *template*. I will outline a general method by which the event builder will work, but most experiments will require the user to tweak the code to fit their individual data set. Additionally I will try to give pointers on how the user can modify the code to better fit other use cases. Hopefully this will allow everyone to work on SPS data from the same basic template!

Some notes about compatibility: The code has been compiled successfully on Scientific Linux 7 and Mac OS X Catalina. There isn't anything here that is specifically os dependent, but you never know. Testing has been done with ROOT ver. 6.14 and 6.20, so anything else is a toss up.

Other than that, I'll get started with how to build the code.

2   Building

Building is done via a makefile located in the top level directory in the project. The user will have to make some minor changes to the makefile to reflect their specific environment. At lines 6 and 7, there is are variables declared as follows:

```
LIBARCHIVE=/usr/local/opt/libarchive/lib/libarchive.dylib
LIBARCHIVE_INCL=/usr/local/opt/libarchive/include
```

These define paths to necessary objects to use `libarchive`, which is a library for reading in compressed and archived file formats in `C++`. The first, `LIBARCHIVE` defines the location and name of the `libarchive` dynamic library, while the second `LIBARCHIVE_INCL`, defines the path to the header files. `libarchive` is typically a member of most os distributions, but it is not always the most up to date version. This code is built assuming that `libarchive` ver. 3 is present, so anything lower will require the user to update. The example shown here is as though `libarchive` was installed via `homebrew` on OS X.

If for some reason you don't want to use `libarchive` or the `binary2root` program, you can of course ignore this part and just remove the `binary2root` as part of the building. To do this go to line 52 where the following is defined:

```
all: $(PCH) $(AEXE) $(MEXE) $(CEXE) $(BEXE)
```

To not build `binary2root` simply delete `$(BEXE)` from this line.

It is important to note that the `libarchive` location, name, and method by which headers can be located will change on a build by build basis. That is, two users that both have Scientific Linux 7 may not necessarily have `libarchive` installed in the same location. User 1 may also be able to ignore the `libarchive` include line, as they placed the header file in their standard search path, while user 2 still needs to add the explicit path call to make.

Building is handled via the `make` command. To build the entire environment simply enter `make` into the command line and watch the output. To build from scratch after an initial build run `make clean` as this will delete all objects and executables, except for one special case. The build uses a precompiled header which contains the commonly used standard library and ROOT library includes (EventBuild.h). This is typically built only once, during the first call of `make` and never needs to be touched again. However, if for some reason you need to clean this as well you can use `make clean_header` to delete the file and force a rebuild.

Note: This makefile is extremely general. Any time a file with a `.cpp` extension is added into one of the `src` program directories it will try to build it. If the user wants to add and additional program/directory, just follow the format outlined here.

3   Data Structure and ROOT dictionary

Now is probably a good time to get into how the data is structured and given to the ROOT environment. To find the data structures, simply open the file `DataStructs.h` in `include`. In here you'll see several defined `C` structs, namely:

- `DPPChannel`

- `DetectorHit`

- `SabreDetector`

- `FPDetector`

- `CoincEvent`

- `ProcessedEvent`

`DPPChannel` is a modified structure of the standard CoMPASS data. The modification is in the typing of the data; CoMPASS reports out in terms of unsigned integers, which can be difficult to manipulate with mathematic operations. Early in the event building, these unsigned quantities are converted to signed counter parts of appropriate length.

`DetectorHit` carries the relevant information of any detector hit: energy (`Long`), timestamp (`Time`), and channel (`Ch`). Note that we use here only the long gated energy; if there is an application which requires using the short gated energy, that should be added into this structure. There are then two detector structures shown here: `SabreDetector` and `FPDetector`. These structures contain combinations of detector hits that are organized by components for each detector array. For example `SabreDetector` has elements for rings and wedges, while the `FPDetector` has elements for the anodes, delay lines, scintillators, and cathode. Note how these are all `std::vector` type quantities. This is important for the way that event building is done. All detector elements should be initialized as `std::vector`s unless you have a very good reason to assume that this detector element will only ever seen ONE hit (including noise) during a coincidence window.

`CoincEvent` is a collection of detectors that should be grouped together in coincidence. This is the quantity which we desire to have the event builder create for us! Here I have an example with an `FPDetector` and an array of 5 `SabreDetectors` (one for each silicon in the physical array).

`ProcessedEvent` is the physics output of the event builder. This is where you would want to define quantities you actually use in real physics analysis. For example, in my version I have defined focal plane positions, angles, and various energies and times. As these are the physical variables, they deserve a little more detail, given below:

- `fp1_tdiff` and `fp2_tdiff` are the time difference in $left - right$ format of either delay line. These are in nanoseconds.

- `fp1_tsum` and `fp2_tsum` are the sum times $left + right$ of the delay lines (ns). This sum should be "consistent" with the total delay of the delay line.

- `fp1_tcheck` and `fp2_tcheck` are the sum times divided by 2, with the anode time subtracted. Essentially, this tells the inherent time resolution of the delay line scheme. There are also some other uses for this parameter, depending on how much you trust the anode time.

- `fp1_y` and `fp2_y` are basically the drift time of each wire, which means they are a measurement of the y position. It should be noted, that after some testing these were found to be of such poor resolution that they were not useful.

- `anodeFront/Back`, `scintLeft/Right`, and `cathode` are all energy values of their respective components.

- `x1` and `x2` are the `tdiff` parameters converted into mm. They are the delay line positions.

- `xavg` is a weighted average of `x1` and `x2` with weights calculated such that `xavg` is the result of a linear interpolation (or extrapolation depending on the scenario) to

3

the kinematic focal plane. If you drew a line with `x1` and `x2`, `xavg` is where that line intersects with the kinematic focal plane.

- `theta` is the incident angle of the particle, calculated by $arctan(\frac{x1-x2}{\text{wire separation}})$

- `sabreRing/Wedge`, `sabreRing/WedgeChannel`, and `sabreRing/WedgeTime` are SABRE data for the largest energy hit in each SABRE detector (one for each detector). These are easy access values for the cleaner.

- `dealyFront/BackLeft/RightE` are energy values from the delay lines themselves. It has been found that in many cases these have higher resolution and less deformation than the anode signals.

- `anodeFront/BackTime` and `scintLeft/RightTime` are the times of the different components (ns).

- `delayFront/BackMaxTime` are the maximum time of the left and right side of the delay lines. These are very useful for estimating the width of the slow coincidence window, as once everything is shifted into place, the delay time is the only remaining time that needs to be accounted for in the window, and even that only needs to be the *maximum* delay.

- `sabreArray` is the full set of SABRE data for the user to handle in the next level of analysis.

In general, the user will be making the most changes to what a `CoincEvent` is and what a `ProcessedEvent` is. These are the parts that really depend on the specifics of each experiment. It should also be noted that for many of these I initialize the structure to unphysical values (negative numbers for energy and time, very large numbers for position). The reason for this is that ROOT TTree fills are not, in general, selective, while most analysis codes are. You define an address for a branch, and then every time you call `MyTree->Fill()` the tree adds an entry to each branch *regardless* of whether or not the value in that branch address has been updated. This means that to make the fill safe, we need to reset the value at the branch address *every time* we go to perform an assignment. This means that if you add member to something like `ProcessedEvent` you should set it to something unphysical which can be easily discarded at a later stage (I will refer to these as dump values).

Now that our data structures are defined in our code, we need to define them for the ROOT environment so we can use them with our TTrees. This is done via ROOT dictionaries. If you are looking for a really detailed description of how this works, I suggest going to the ROOT user guide for more information, as there is a lot of black box type behavior here. ROOT installations come with a dictionary generator `rootcling` which does pretty much all of the heavy lifting for us. To use `rootcling` we need to have a file that

defines which structures need to be added to the dictionary. This is called the `LinkDef.h` file, and for us this file is called `LinkDef_sps.h` (`rootcling` can only accept header files as arguments). The `LinkDef` file contains a whole bunch of preprocessor directives. First it checks to make sure cling is well defined, and then it links all of the structures we want. Note that here I've used structures because our data is pretty simple, but you can link more complicated things like classes if you need. The building of the actual dictionary is handled by the makefile. At lines 35 through 37 the dictionary variables are defined (`$LIB` defines the compiled object the dictionary makes). Lines 73 and 74 define the command that actually builds the dictionary, and then lines 69 through 71 outline the compilation of the dictionary into an object file. Dictionary generation also makes a .pcm file, which is used to help the dictionary function at runtime. The .pcm must be in the same directory as the executable (which is done at line 71 of the makefile).

4   The Event Builder Program

Currently the event building program is called `analyzer`, and it has an analysis pipeline as shown in Fig. 1 (blue text indicates the program option which leads to that particular branching path). There are four options for running the program:
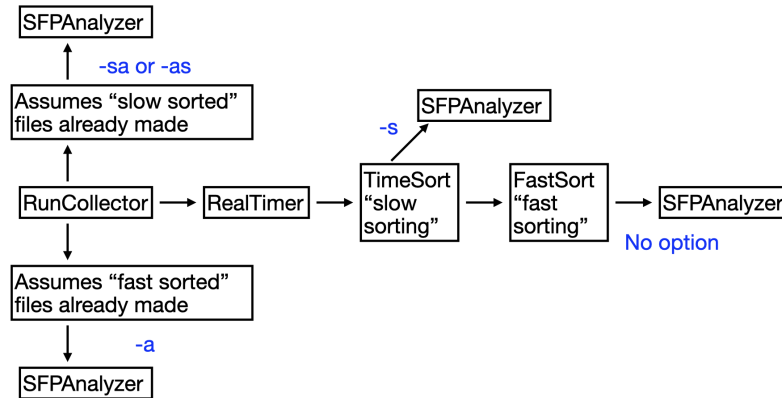

Figure 1: Flow chart of analyzer program. Blue text indicates which program option leads to which branching path

- `-s`: this runs only the "slow sorting". It passes information to `TimeSort` and then the `SFPAnalyzer`

- `-a`: this runs only the "SFPAnalyzer". It skips all stages of analysis assuming that fast sorted files are already created.

5

- **-sa** or **-as**: this is the same as **-a** except that it assumes slow sorted files are already created.

- No option: this runs the entire pipeline.

Additionally, the analyzer requires an input file, of which examples are given in the repository. The idea of the pipeline is as follows:

1. Shift all of the data so that all detectors occur at the same timestamp. This is done by adding or subtracting values from the timestamps so that all detectors are essentially coincident. In general this means pushing the scintillator and any other separate array (such as SABRE) to match the timestamp of a specific part of the focal plane (typically an anode). This allows us to minimize the width of the coincidence window, which ideally should just be the separation in time from the anode to the maximum delay time. This is handled via the `RealTimer` class.

2. Next we grab all of the data in the "slow" coincidence window (described in the previous step). The idea is that the slow window be as general as possible. This is just the max amount of time we are willing to look for data that should be lumped together into a single event structure. This is done in the `TimeSort` class.

3. Now we break down that slow event into individual "fast" events. The goal is to pick out which parts of that slow event actually are correlated in time. Basically you want to reject extraneous hits and properly group the "good" hits. This is the part of the event building that will in general change from experiment to experiment. This depends heavily on which detectors are involved, and the settings of each specific detector. This is done with the `FastSort` class.

4. Finally, now that we have our fast events, we want to leave detector parameters and transform to physics parameters. This is done in basic form with the class `SFPAnalyzer`. In general this part will also have to change from experiment to experiment, but this should be kept fairly simple at this stage. The goal with this is merely to produce enough parameters to ensure that the event builder is working correctly, and that very preliminary analysis can be performed.

NOTE: All of the input files described below are *whitespace* delineated. That is, the white space determines the location of the next variable. So when editing, be *sure* to put spaces in between parameters and their names!

4.1   Analyzer inputs

Here is an example of what the analyzer expects as inputs:

```
1  Ztarget: 5 Atarget: 10 Zproj: 2 Aproj: 3 Zeject: 2 Aeject: 4
2  BeamE(MeV): 24 Angle(deg): 20 Bfield(G): 9500
3  Board_shift_file: etc/ShiftMap_April2020.txt
4  Scint_offset(ps): 0.65e6
5  coincidence_window(ps): 1.5e6
6  si_fast_coincidence_window(ps): 0.125e6
7  ion_chamber_fast_coincidence_window(ps): 0.25e6
8  DataDir: /Volumes/Lacie/9BMarch2020/raw_root/
9  MinRun: 99 MaxRun: 99
10 TimeshiftedDir: /Volumes/Lacie/9BMarch2020/shifted/
11 SortedDir: /Volumes/Lacie/9BMarch2020/sorted/
12 FastDir: /Volumes/Lacie/9BMarch2020/fast/
13 AnalyzedDir: /Volumes/Lacie/9BMarch2020/analyzed/
14 SABREChannelMapFile: ./etc/ChannelMap_March2020.dat
15 GainMatchingFile: ./etc/March2020_gainmatch_2.0V_5486Am241.txt
```

Lines 1 and 2 contain the kinematic parameters necessary to perform kinematic corrections with x-avg. Line 3 has the path to a `.txt` file which contains board by board time shifts in picoseconds (for something like SABRE). Line 4 has the shift for the scintillator in picoseconds. Lines 5, 6, and 7 contain the various coincidence windows used in the current setup. Lines 8 and 9 contain information on where the raw data is stored. The analyzer expects ROOT data format. A run number range `[MinRun, MaxRun]` can be specified. Alternatively, if one of either `MinRun` or `MaxRun` is set to a negative value, the analyzer takes all files with a `.root` format in the data directory. Lines 10, 11, 12, and 13 specify locations to which data will be written after each stage in the pipeline. Names for files are automatically generated using a run number.

NOTE: The name format is important! The program expects all files to have the following name style: `run_<number>.<extension>` Any format other than this will require editing of both the `main.cpp` and `RunCollector.cpp`. This is true for *every* part of this environment. Probably easier to just rename files.

Line 14 specifies a channel map file. Currently the channel map file is used to specify all channels that *are not* part of the focal plane detector. In the example map, the channel map specifies all SABRE detector channels. The focal plane is mapped in an `enum` in `TimeSort.h`, though hopefully these channels change very infrequently. Line 15 gives the path to a gain matching file. For many types of detectors it is useful to gain match channels. Currently gain matching is set up for SABRE, but with minor changes to `GainMatch.cpp` this could be modified for other arrays as well. To make changes to the input file, one needs only to adjust parts of the `analyzer main.cpp`. The relevant area is shown below:

```
1  int main(int argc, char *argv[]) {
2    if(argc == 2 || argc == 3) {
3
4      int c;
5      while((c=getopt(argc,argv,optString)) != -1) {
```

```
 6        switch(c) {
 7          case 's':
 8            options.slowFlag = true;
 9            break;
10          case 'a':
11            options.onlyAnalyzeFlag = true;
12            break;
13        }
14      }
15
16      TApplication app("app", &argc, argv);
17      char* name;
18      if(app.Argc() == 2) name = app.Argv(1);
19      else   name = app.Argv(2);
20      ifstream input(name);
21      if(input.is_open()) {
22        string dir, shifted, sorted, analyzed, junk, fast, map, gains, shifts
     ;
23        int zt, at, zp, ap, ze, ae;
24        double ep, angle, b;
25        float scint, cw, si_fcw, ion_fcw;
26        int min, max;
27        input >> junk >> zt >> junk >> at >> junk >> zp >> junk >> ap >> junk >> ze >> junk >> ae;
28        input >> junk >> ep >> junk >> angle >> junk >> b;
29        input >> junk >> shifts >> junk >> scint >> junk >> cw >> junk >> si_fcw >> junk >>
     ion_fcw;
30        input >> junk >> dir >> junk >> min >> junk >> max;
31        input >> junk >> shifted >> junk >> sorted >> junk >> fast >> junk >> analyzed;
32        input >> junk >> map;
33        input >> junk >> gains;
34        input.close();
```

Lines 23-34 are where inputs are obtained. Adding more is as simple as creating a new variable and placing the **>>** operators in the right place.

4.2 Analyzer Standard Use

The recommended use sequence is as follows:

1. Run the `analyzer` with `-s` and a wide coincidence window (usually 3.0 μs will do the job) to determine how to place shifts for the scintillator and other components.

2. Run with no options specified and a tighter slow gate, with wide fast gates to determine the fast gates.

3. Run with no options now with all gates narrowed down

4. Tweak as needed

4.3 Cleaner

The `cleaner` is a small program which generates a whole mess of histograms over a large range of data files. It is to be used on final analyzed files from the `analyzer`, and as such changes to the `analyzer` should be reflected in the `cleaner`. Also, it is important to note that the cleaner can access `TCutG` type objects saved to ROOT files. First let's look at an example of the input file for the cleaner:

```
1 DataDir: /Volumes/Lacie/9BMarch2020/analyzed/
2 MinRun: 9 MaxRun: -1
3 HistogramFile: /Volumes/Lacie/9BMarch2020/9b_histograms/9
      b_spectra_fulldata_including_garbage_testing_fasttime_fp_scint0.65
      _indibds_4real.root
4 E_dE_CutFile: /Volumes/Lacie/9BMarch2020/cuts/alphaCut_dbr_sl.root
5 x1_x2_CutFile: /Volumes/Lacie/9BMarch2020/cuts/x1x2Cut.root
6 E_positon_CutFile: /Volumes/Lacie/9BMarch2020/cuts/scintXavgCut.root
7 dE_position_CutFile: /Volumes/Lacie/9BMarch2020/cuts/delayEXavgCut.root
```

Line one specifies the location of analyzed data files, and line 2 specifies the run number range for the `cleaner` in the same way as the `analyzer` (inclusive, negative number indicates run all files in data directory). Line 3 gives the full path name of the histogram file that will be created by the `cleaner`. The final four lines are full path names of cut files. You can see that in this example we have included some of the basic focal plane gates. Each of these cuts will be saved to the histogram file, so that they can be referenced again later. For the input file, adding a new cut is as simple as adding a new line in a similar fashion to those shown above. Handling the input read in is very similar to the `analyzer`; add a new temporary variable in the `cleaner main.cpp` and pass it along to the appropriate function, `SFPCleaner::SetCuts()`. To make additional cuts, you'll have to modify the `SFPCleaner::SetCuts()` function to take an extra argument and then properly handle that argument. Here is a snippet showing the standard method of `SFPCleaner::SetCuts()`:

```
1 int SFPCleaner::SetCuts(string edename, string dexname, string exname,
      string xxname) {
2   edefile = new TFile(edename.c_str(), "READ");
3   if(edefile->IsOpen()) {
4     EdECut = (TCutG*) edefile->Get("CUTG");
5     EdECut->SetName("EdECut");
6     rootObj->Add(EdECut);
7   }
8   dexfile = new TFile(dexname.c_str(), "READ");
9   if(dexfile->IsOpen()) {
10    dExCut = (TCutG*) dexfile->Get("CUTG");
11    dExCut->SetName("dExCut");
12    rootObj->Add(dExCut);
13  }
14  exfile = new TFile(exname.c_str(), "READ");
15  if(exfile->IsOpen()) {
```

```
16     ExCut = (TCutG*) exfile->Get("CUTG");
17     ExCut->SetName("ExCut");
18     rootObj->Add(ExCut);
19   }
20   xxfile = new TFile(xxname.c_str(), "READ");
21   if(xxfile->IsOpen()) {
22     x1x2Cut = (TCutG*) xxfile->Get("CUTG");
23     x1x2Cut->SetName("x1x2Cut");
24     rootObj->Add(x1x2Cut);
25   }
26   if(EdECut != NULL && dExCut != NULL && x1x2Cut != NULL && ExCut != NULL)
      {
27     return 1;
28   } else {
29     return 0;
30   }
31 }
```

A couple of things of note: each `TCutG` and `TFile` are declared as class globals. This is important! If the `TFile` is closed, it deallocates all memory associated with it, including our `TCutG`. To handle this, the `TFile*` variable is set to `NULL` in the constructor and closing is handled in the destructor as shown below:

```
1 SFPCleaner::~SFPCleaner() {
2   delete event_address;
3   if(edefile != NULL && edefile->IsOpen()) edefile->Close();
4   if(dexfile != NULL && dexfile->IsOpen()) dexfile->Close();
5   if(exfile != NULL && exfile->IsOpen()) exfile->Close();
6   if(xxfile != NULL && xxfile->IsOpen()) xxfile->Close();
7 }
```

You can see that we first check that the `TFile` pointer is set, and then check if it is open before deallocating. It should also be noted that `cleaner` takes advantage of a class called `TChain` which has the assumption that the `TTree`s in all of the files you pass have the same name. If you use the `analyzer` this will always be the case, but just in case changes are necessary, I want this to be clear.

4.4   Binary conversion

Converting binary is a fairly simple process. The input file is very similar to the `analyzer` and `cleaner` inputs. It asks for a data directory, looks for files within a specified run number range, and outputs a new `.root` file for each binary file. There is one restriction as currently implemented: It expects compressed archives of the full run, not each individual channel binary file that CoMPASS outputs. This is easy to do though; simply run `tar -cvzf run_<run number>.targ.gz  <binary directory>*.bin` to compress and archive them all. `binary2root` then opens the archive and converts the entire archive into a `.root` file. Currently it is implemented to *only* accept `.tar.gz` format archives, but in

principle `libarchive` can handle many formats, I just don't see the point of allowing that much freedom, as it makes `RunCollector` much less stable.

4.5 Merging

After completing the full event building process, one will probably want to take their data and do all kinds of fun other stuff to it. Instead of copying and loading around hundreds of analyzed files, a convenient program called `merger` has been made which will take a whole bunch of files and mush them together. There are two merging methods: one is via `TChain`'s built in `Merge()` function, while the second is a call to `hadd`. Documentation for both of these methods can be found in the ROOT documentation. For us these methods are virtually identical; the only real reason to use one over the other is if you need to merge histograms as well as trees, as in this case you would need to use `hadd`. Input file format is the usual story: data directory, run range, and output file location.

4.6 Processing

For convenience, I've included a little shell wrapper script that runs the usual sequence of programs:

```zsh
#!/bin/zsh

./bin/binary2root binary2root_input.txt
./bin/analyzer analyzer_input.txt
./bin/cleaner cleaner_input.txt
```

Feel free to make use of this as a way to avoid constantly calling to bin by hand. (This was made on a Mac running OS X Catalina where the shell is `zsh`, not `bash`. If you don't run `zsh`, just change the shebang to `#!/bin/bash`)

5 Example

Ok now that we've gone over the basics, lets look at an example of what running this would actually look like. In the repository I've included a directory called `example` which contains both this document and an example run called `run_75`. This is $^{12}C(^{3}He, \alpha)^{11}C$ data from an experiment that was run in March 2020.

Let's set our input files up. First lets do the `binary2root` input file. Use your favorite text editor and open `binary2root_input.txt` and set it to look like this:

```
DataFile: <your_path>/GWM_EventBuilder/example/raw_binary/
MinRun: -99 MaxRun: 99
OutputFile: <your_path>/GWM_EventBuilder/example/raw_root/
```

`<your_path>` is just a place holder. This should be your explicit full path to where ever the distribuition is on your machine, which you can pop by using the command `pwd` at the

command line. And here we've set it to pull all runs in the example directory by setting `MinRun` to a negative number. But we could've just as easily set `MinRun: 75 MaxRun: 75` since we know there is only one run (75) in the directory.

Next, `analyzer_input.txt`:

```
1  Ztarget: 6 Atarget: 12 Zproj: 2 Aproj: 3 Zeject: 2 Aeject: 4
2  BeamE(MeV): 24 Angle(deg): 20 Bfield(G): 9500
3  Board_shift_file: none
4  Scint_offset(ps): 0.0
5  coincidence_window(ps): 3e6
6  si_fast_coincidence_window(ps): 3.0e6
7  ion_chamber_fast_coincidence_window(ps): 3.0e6
8  DataDir: <your_path>/GWM_EventBuilder/example/raw_root/
9  MinRun: -99 MaxRun: 99
10 TimeshiftedDir: <your_path>/GWM_EventBuilder/example/shifted/
11 SortedDir: <your_path>/GWM_EventBuilder/example//sorted/
12 FastDir: <your_path/GWM_EventBuilder/example/fast/
13 AnalyzedDir: <your_path>/GWM_EventBuilder/example/analyzed/
14 SABREChannelMapFile: ./etc/ChannelMap_March2020.dat
15 GainMatchingFile: ./etc/March2020_gainmatch_2.0V_5486Am241.txt
```

Here we've done a couple of things. First, we've set the kinematic parameters to be tuned for our reaction. Second, we've set the paths for our files similarly to `binary2root`. And then we've put in preliminary values for shifts and windows. This is the trickiest part of the whole thing. We've set `none` as our board shift file, which is fine. This just applies a shift of 0 to all boards (does nothing). Next we set the scint offset to 0 as well. Finally we set all windows to 3 μs. A 3 μs window is pretty wide for this setup, and we'll see that shortly. We've also passed default channel map and gain matching files that are included with the repo. Now we're ready for our first pass of running. So lets modify `process` to do only `binary2root` and `analyzer -s` as shown:

```
1  #!/bin/zsh
2
3  ./bin/binary2root binary2root_input.txt
4  ./bin/analyzer -s analyzer_input.txt
5  #./bin/cleaner cleaner_input.txt
```

We've commented out `cleaner` as we don't have cuts made yet to give it. And we've put in the `-s` option as we're not ready to make fast coincidences. So lets run this puppy (just do `./process` on the command line). Aaaaand you should see a ton of output barfed to your terminal, looking like as follows:

```
1  ---------binary2root---------
2  Name of data dir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      raw_binary/
3  Name of output dir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      raw_root/
4  Searching directory: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      raw_binary/ for files starting with:  and ending with: .tar.gz
```

```
 5 Found file: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/raw_binary
     /run_75.tar.gz
 6 Working on /Users/gordonmccann/Desktop/GWM_EventBuilder/example/raw_binary/
     run_75.tar.gz...
 7 Succesfully opened archive: /Users/gordonmccann/Desktop/GWM_EventBuilder/
     example/raw_binary/run_75.tar.gz
 8 Beginning unpacking...
 9 Number of files in archive: 1
10 Parsing next file in archive: CH0@V1725_324_Data_run_75.bin
11 Parsing next file in archive: CH0@V1725_325_Data_run_75.bin
12 Parsing next file in archive: CH0@V1725_334_Data_run_75.bin
13 Parsing next file in archive: CH0@V1725_336_Data_run_75.bin
14 Parsing next file in archive: CH0@V1725_379_Data_run_75.bin
15 Parsing next file in archive: CH0@V1725_405_Data_run_75.bin
16 Parsing next file in archive: CH0@V1730_82_Data_run_75.bin
17 Parsing next file in archive: CH0@V1730_89_Data_run_75.bin
18 Parsing next file in archive: CH10@V1725_323_Data_run_75.bin
19 Parsing next file in archive: CH10@V1725_324_Data_run_75.bin
20 Parsing next file in archive: CH10@V1725_325_Data_run_75.bin
21 Parsing next file in archive: CH10@V1725_334_Data_run_75.bin
22 Parsing next file in archive: CH10@V1725_336_Data_run_75.bin
23 Parsing next file in archive: CH10@V1725_379_Data_run_75.bin
24 Parsing next file in archive: CH10@V1725_405_Data_run_75.bin
25 Parsing next file in archive: CH10@V1730_82_Data_run_75.bin
26 Parsing next file in archive: CH11@V1725_323_Data_run_75.bin
27 Parsing next file in archive: CH11@V1725_324_Data_run_75.bin
28 Parsing next file in archive: CH11@V1725_325_Data_run_75.bin
29 Parsing next file in archive: CH11@V1725_334_Data_run_75.bin
30 Parsing next file in archive: CH11@V1725_336_Data_run_75.bin
31 Parsing next file in archive: CH11@V1725_379_Data_run_75.bin
32 Parsing next file in archive: CH11@V1725_405_Data_run_75.bin
33 Parsing next file in archive: CH11@V1730_82_Data_run_75.bin
34 Parsing next file in archive: CH12@V1725_323_Data_run_75.bin
35 Parsing next file in archive: CH12@V1725_324_Data_run_75.bin
36 Parsing next file in archive: CH12@V1725_325_Data_run_75.bin
37 Parsing next file in archive: CH12@V1725_334_Data_run_75.bin
38 Parsing next file in archive: CH12@V1725_336_Data_run_75.bin
39 Parsing next file in archive: CH12@V1725_379_Data_run_75.bin
40 Parsing next file in archive: CH12@V1725_405_Data_run_75.bin
41 Parsing next file in archive: CH13@V1725_323_Data_run_75.bin
42 Parsing next file in archive: CH13@V1725_324_Data_run_75.bin
43 Parsing next file in archive: CH13@V1725_325_Data_run_75.bin
44 Parsing next file in archive: CH13@V1725_334_Data_run_75.bin
45 Parsing next file in archive: CH13@V1725_336_Data_run_75.bin
46 Parsing next file in archive: CH13@V1725_379_Data_run_75.bin
47 Parsing next file in archive: CH13@V1725_405_Data_run_75.bin
48 Parsing next file in archive: CH13@V1730_82_Data_run_75.bin
49 Parsing next file in archive: CH14@V1725_323_Data_run_75.bin
50 Parsing next file in archive: CH14@V1725_324_Data_run_75.bin
51 Parsing next file in archive: CH14@V1725_325_Data_run_75.bin
```

```
52 Parsing next file in archive: CH14@V1725_334_Data_run_75.bin
53 Parsing next file in archive: CH14@V1725_336_Data_run_75.bin
54 Parsing next file in archive: CH14@V1725_379_Data_run_75.bin
55 Parsing next file in archive: CH14@V1725_405_Data_run_75.bin
56 Parsing next file in archive: CH15@V1725_323_Data_run_75.bin
57 Parsing next file in archive: CH15@V1725_324_Data_run_75.bin
58 Parsing next file in archive: CH15@V1725_325_Data_run_75.bin
59 Parsing next file in archive: CH15@V1725_334_Data_run_75.bin
60 Parsing next file in archive: CH15@V1725_336_Data_run_75.bin
61 Parsing next file in archive: CH15@V1725_379_Data_run_75.bin
62 Parsing next file in archive: CH15@V1725_405_Data_run_75.bin
63 Parsing next file in archive: CH15@V1730_82_Data_run_75.bin
64 Parsing next file in archive: CH1@V1725_323_Data_run_75.bin
65 Parsing next file in archive: CH1@V1725_324_Data_run_75.bin
66 Parsing next file in archive: CH1@V1725_325_Data_run_75.bin
67 Parsing next file in archive: CH1@V1725_334_Data_run_75.bin
68 Parsing next file in archive: CH1@V1725_336_Data_run_75.bin
69 Parsing next file in archive: CH1@V1725_379_Data_run_75.bin
70 Parsing next file in archive: CH1@V1725_405_Data_run_75.bin
71 Parsing next file in archive: CH1@V1730_82_Data_run_75.bin
72 Parsing next file in archive: CH1@V1730_89_Data_run_75.bin
73 Parsing next file in archive: CH2@V1725_323_Data_run_75.bin
74 Parsing next file in archive: CH2@V1725_325_Data_run_75.bin
75 Parsing next file in archive: CH2@V1725_334_Data_run_75.bin
76 Parsing next file in archive: CH2@V1725_336_Data_run_75.bin
77 Parsing next file in archive: CH2@V1725_379_Data_run_75.bin
78 Parsing next file in archive: CH2@V1725_405_Data_run_75.bin
79 Parsing next file in archive: CH2@V1730_89_Data_run_75.bin
80 Parsing next file in archive: CH3@V1725_323_Data_run_75.bin
81 Parsing next file in archive: CH3@V1725_324_Data_run_75.bin
82 Parsing next file in archive: CH3@V1725_325_Data_run_75.bin
83 Parsing next file in archive: CH3@V1725_334_Data_run_75.bin
84 Parsing next file in archive: CH3@V1725_336_Data_run_75.bin
85 Parsing next file in archive: CH3@V1725_379_Data_run_75.bin
86 Parsing next file in archive: CH3@V1725_405_Data_run_75.bin
87 Parsing next file in archive: CH3@V1730_89_Data_run_75.bin
88 Parsing next file in archive: CH4@V1725_323_Data_run_75.bin
89 Parsing next file in archive: CH4@V1725_324_Data_run_75.bin
90 Parsing next file in archive: CH4@V1725_325_Data_run_75.bin
91 Parsing next file in archive: CH4@V1725_334_Data_run_75.bin
92 Parsing next file in archive: CH4@V1725_336_Data_run_75.bin
93 Parsing next file in archive: CH4@V1725_379_Data_run_75.bin
94 Parsing next file in archive: CH4@V1725_405_Data_run_75.bin
95 Parsing next file in archive: CH4@V1730_89_Data_run_75.bin
96 Parsing next file in archive: CH5@V1725_323_Data_run_75.bin
97 Parsing next file in archive: CH5@V1725_324_Data_run_75.bin
98 Parsing next file in archive: CH5@V1725_325_Data_run_75.bin
99 Parsing next file in archive: CH5@V1725_334_Data_run_75.bin
100 Parsing next file in archive: CH5@V1725_336_Data_run_75.bin
101 Parsing next file in archive: CH5@V1725_379_Data_run_75.bin
```

14

```
102 Parsing next file in archive: CH5@V1725_405_Data_run_75.bin
103 Parsing next file in archive: CH5@V1730_89_Data_run_75.bin
104 Parsing next file in archive: CH6@V1725_323_Data_run_75.bin
105 Parsing next file in archive: CH6@V1725_324_Data_run_75.bin
106 Parsing next file in archive: CH6@V1725_325_Data_run_75.bin
107 Parsing next file in archive: CH6@V1725_334_Data_run_75.bin
108 Parsing next file in archive: CH6@V1725_336_Data_run_75.bin
109 Parsing next file in archive: CH6@V1725_379_Data_run_75.bin
110 Parsing next file in archive: CH6@V1725_405_Data_run_75.bin
111 Parsing next file in archive: CH6@V1730_89_Data_run_75.bin
112 Parsing next file in archive: CH7@V1725_323_Data_run_75.bin
113 Parsing next file in archive: CH7@V1725_324_Data_run_75.bin
114 Parsing next file in archive: CH7@V1725_325_Data_run_75.bin
115 Parsing next file in archive: CH7@V1725_334_Data_run_75.bin
116 Parsing next file in archive: CH7@V1725_336_Data_run_75.bin
117 Parsing next file in archive: CH7@V1725_379_Data_run_75.bin
118 Parsing next file in archive: CH7@V1725_405_Data_run_75.bin
119 Parsing next file in archive: CH7@V1730_82_Data_run_75.bin
120 Parsing next file in archive: CH7@V1730_89_Data_run_75.bin
121 Parsing next file in archive: CH8@V1725_323_Data_run_75.bin
122 Parsing next file in archive: CH8@V1725_324_Data_run_75.bin
123 Parsing next file in archive: CH8@V1725_325_Data_run_75.bin
124 Parsing next file in archive: CH8@V1725_334_Data_run_75.bin
125 Parsing next file in archive: CH8@V1725_336_Data_run_75.bin
126 Parsing next file in archive: CH8@V1725_379_Data_run_75.bin
127 Parsing next file in archive: CH8@V1725_405_Data_run_75.bin
128 Parsing next file in archive: CH8@V1730_82_Data_run_75.bin
129 Parsing next file in archive: CH9@V1725_323_Data_run_75.bin
130 Parsing next file in archive: CH9@V1725_324_Data_run_75.bin
131 Parsing next file in archive: CH9@V1725_325_Data_run_75.bin
132 Parsing next file in archive: CH9@V1725_334_Data_run_75.bin
133 Parsing next file in archive: CH9@V1725_336_Data_run_75.bin
134 Parsing next file in archive: CH9@V1725_379_Data_run_75.bin
135 Parsing next file in archive: CH9@V1725_405_Data_run_75.bin
136 Parsing next file in archive: CH9@V1730_82_Data_run_75.bin
137 Reached end of archive.
138 -----------------------------
139 ----------------------------------------------------
140 |~~~~~~~~~~~~ GWM SPS-SABRE Analyzer ~~~~~~~~~~~~|
141                                          _
142                                         | |
143                                         | |
144    ___   _ __   ___   ___    ____ _| |__   _ _____
145   / __| | '  \/ __|/ __| / __ ` |   _ \| '__/ __ \
146   \__ \| |\  \__ \\__ \| |   |   | | | | | | | |__| |
147    __) | |/  /__) |__) | |__|   | |_| | | | |   ___/
148   |___/| .__/|___/____/ \____,_|____/|_|   \____|
149        | |
150        | |
151        |_|
```

15

```
152
153 ----------------------------------------------------
154 ~~~~~~~~~~~~~~~~~ Run Information ~~~~~~~~~~~~~~~~~
155 Data Directory: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      raw_root/
156 Timeshifted File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      shifted/run*.root
157 Sorted File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example//sorted/
      run*.root
158 Fast Coincidence File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example
      /fast/run*.root
159 Analyzed File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      analyzed/run*.root
160 Zt: 6 At: 12 Ze: 2 Ae: 4 Zp: 2 Ap: 3 E: 24 angle: 20 B: 9500
161 scintillator offset: 0
162 Board offset file: none
163 Coincidence window: 3e+06
164 Si Fast Coincidence Window: 3e+06
165 Ion Chamber Fast Coincidence Window: 3e+06
166 SABRE Channel Map File: ./etc/ChannelMap_March2020.dat
167 Gain Matching File: ./etc/March2020_gainmatch_2.0V_5486Am241.txt
168 ----------------------------------------------------
169 ~~~~~~~~~~~~~~~~~ Analyzer Output ~~~~~~~~~~~~~~~~~
170 Searching directory: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      raw_root/ for files starting with: compass and ending with: .root
171 Found file: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/raw_root/
      compass_run_75.root
172 ----------------------------------------------------
173 Working on /Users/gordonmccann/Desktop/GWM_EventBuilder/example/raw_root/
      compass_run_75.root...
174 Unable to open shift input file!
175 Locking shift map (all shifts = 0).
176 K = 0.0991994
177 w1: 1.83619 w2: -0.836193
178 Shifting the timestamps of SABRE and the scintillator...
179 File is 50% complete
180 Timestamps succesfully shifted
181 Sorting the file by timestamp...
182 Percent of file processed: 50.00000%
183 Only slow sorting option passed
184 Skipping fast sort... Performing basic analysis...
185 Percent of file processed: 50.00%
186 ----------------------------------------------------
```

The part under the `binary2root` header (lines 1-138) show that we have successfully found and opened an archive. It then lists every file that it finds in the archive (you can check and see that everything is there). Next is the `analyzer` (lines 139-186). It outputs a splash (deal with it) and then a whole mess of information that you told it, so that you can double check your inputs. Under the `Analyzer Output` header is the information from

running. You can see that it locked our shifts to 0, calculated the kinematic weights for xavg, performed shifting and slow sorting, skipped fast sorting, and did the basic analysis. Now lets make some cuts so we can run the cleaner.

Open the analyzed file and lets make some plots. There are default made histograms, but these aren't always best to use. First let's do particle ID through E-dE. Run the following command to the ROOT interpreter with the analyzed file open:

```
root [2] SPSTree->Draw("delayBackRightE:scintLeft>>TH2F(512,0,4096,512,0,4096)","","colz")
```

and you should see the plot shown in Fig 2. Now lets gate on the αs, as shown in Fig 3.
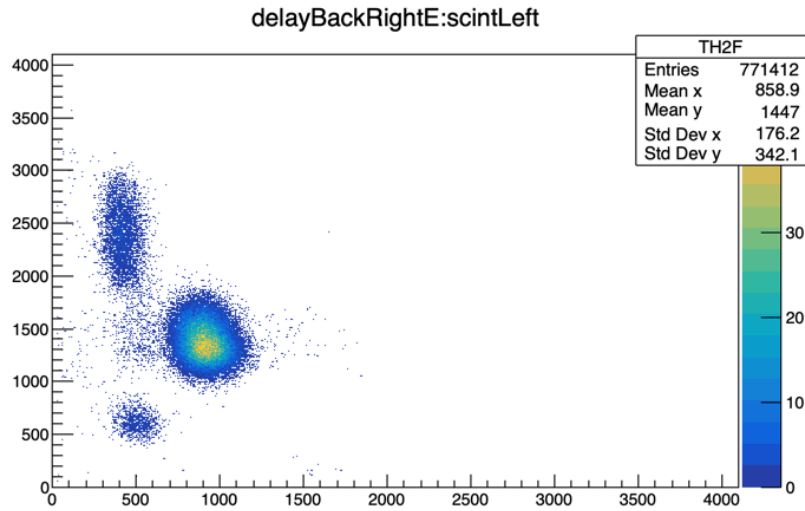
Figure 2: Back right delay energy (dE) vs. left scintillator energy (E), one of the common particle ID plots

Since we're not done with event building rough cuts at this stage are fine. Now you can save this cut to a file by right clicking it and selecting the option `SaveAs`. Give its file a name, and then move the file to `./example/cuts/`. Below are example plots (Fig's 4) for all of the other cuts in the default `cleaner` input.

Ok so now that we've made our cuts, lets update the `cleaner_input.txt` as shown:

```
1  DataDir: <your_path>/GWM_EventBuilder/example/analyzed/
2  MinRun: 9 MaxRun: -1
3  HistogramFile: <your_path>/GWM_EventBuilder/example/histograms/<
      histofile_name>.root
4  E_dE_CutFile: <your_path>/GWM_EventBuilder/example/cuts/<edecut_name>.root
5  x1_x2_CutFile: <your_path>/GWM_EventBuilder/example/cuts/<x1x2cut_name>.
      root
6  E_positon_CutFile: <your_path>/GWM_EventBuilder/example/cuts/<exavgcut_name
      >.root
```
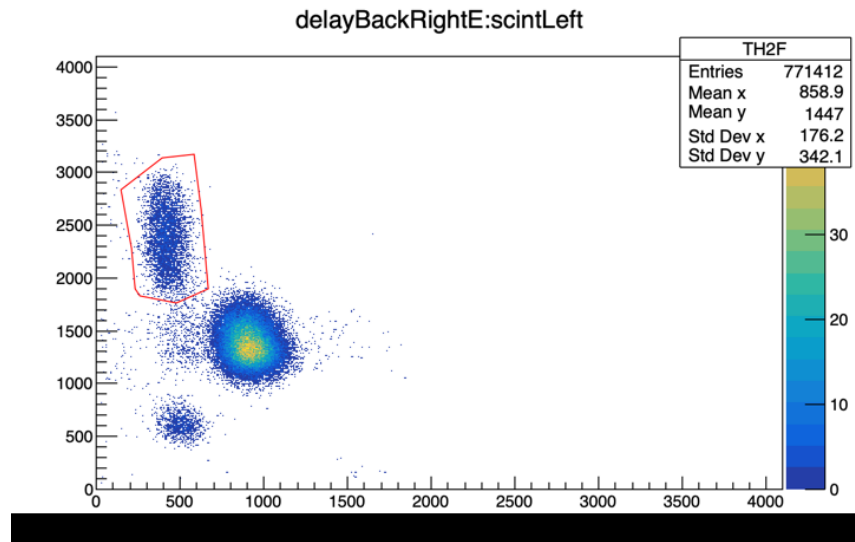
Figure 3: E-dE plot with a gate on α particles

```
7 dE_position_CutFile: <your_path >/GWM_EventBuilder/example/cuts/<
      dexavgcut_name >root
```

Here again replace the `<>` values with specifics of your system. Now we can run the cleaner to generate our first histogram file; do this via the command `./bin/cleaner cleaner_input.txt`. You should see an output that looks like this:

```
1 ------ SPS-SABRE Histogrammer & Cleaner ------
2 Data Dir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/analyzed/
3 Histogram File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      histograms/12C_histo.root
4 E-dE Cut File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/cuts/
      ede_alphaCut.root
5 E-xavg Cut File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/cuts/
      ex_alphaCut.root
6 dE-xavg Cut File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/cuts
      /dex_alphaCut.root
7 x1-x2 Cut File: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/cuts/
      x1x2Cut.root
8 Running cleaner...
9 Searching directory: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
      analyzed/ for files starting with:  and ending with: .root
10 Found file: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/analyzed/
      run_75.root
11 Total number of events: 771412
12 Percent of file processed: 50%
13 Cleaner complete
14 ----------------------------------------------
```
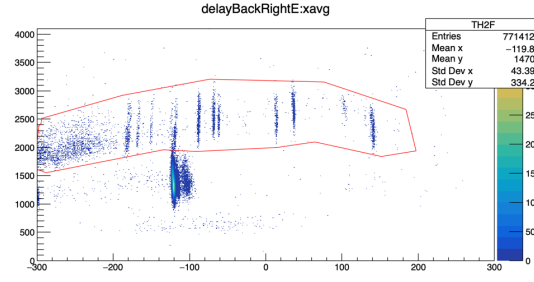
(a) dE vs. xavg
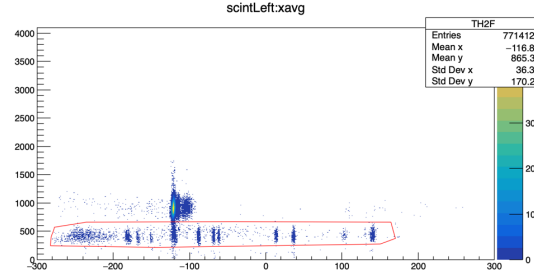
(b) E vs. xavg
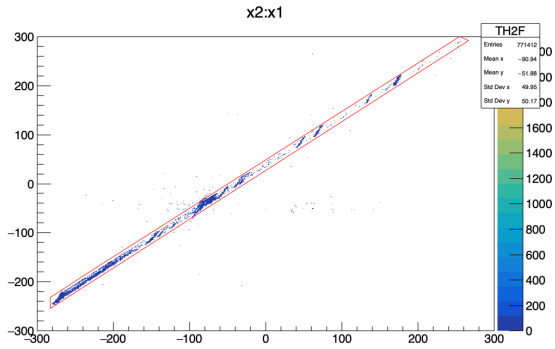
(c) x1 vs. x2

Figure 4: Three example histograms where a) and b) are gated on αs and c) is gated on correlation

First it shows us what it interpreted our inputs as, and then it tells you which files you found, the total number of events found, and that it finished successfully. Now if we open up our histogram file, we can begin to determine our shifts. First we'll determine the shift fo the scintillator to line it up with the back anode. Open up the plot labeled `anodeRelBackTime_toScint` and you should see the following plot Fig 5 We want to center the big peaked structure around 0 (anode back time minus scint left time is 0). Its clear
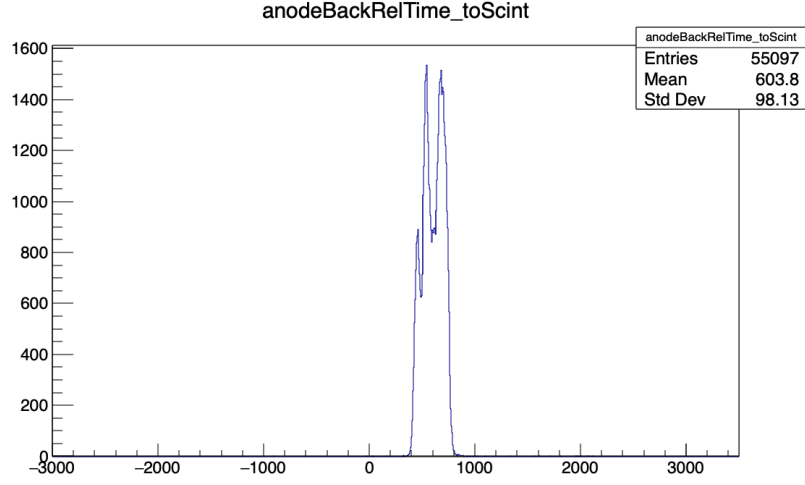
Figure 5: Plot of back anode timestamp minus left scintillator timestamp in nanoseconds

that we need to shift the scint about 0.65 μs. Now, this experiment used SABRE, but ^{11}C is stable for our energy range, so there is very little data in SABRE. If you were to look for SABRE shifts you would use the plots `sabreRelRTScint_sabreRingChannel` and `sabreRelWTScint_sabreWedgeChannel` to determine board by board shifts. I've included the plots (Fig 6) from this data set as an example, and we'll shift them by the shifts that I know apply to this data set anyways so you can get the picture. Note that in this method


(a) SABRE rings

(b) SABRE wedges
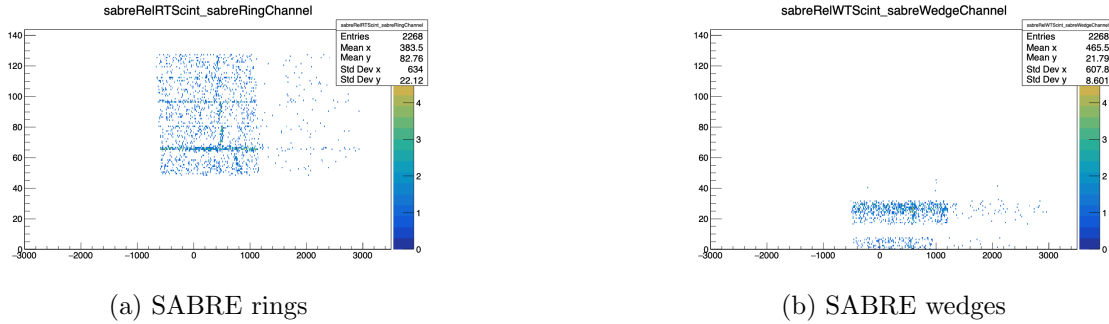
Figure 6: SABRE timestamps minus left scintillator timestamps; y-axis is global channel number ((Board)16 + Channel)

SABRE shifts are determined relative to scintillator shifts! This means that the shift you see that needs applied in the above SABRE plots needs to have the scint shift subtracted from it for you to use it. Before we move on, lets take one look at what our final data looks like in `xavg_bothplanes_edecut` and `xavg_edecut_sabrefcoinc` histogram (Fig. 7). It
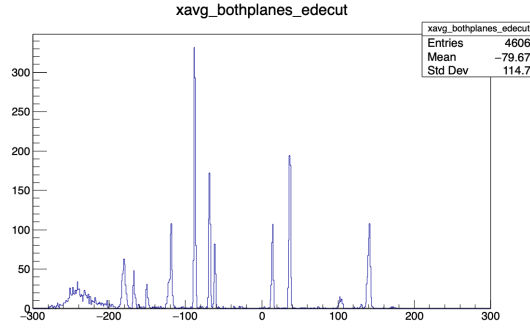
(a) xavg gated on αs
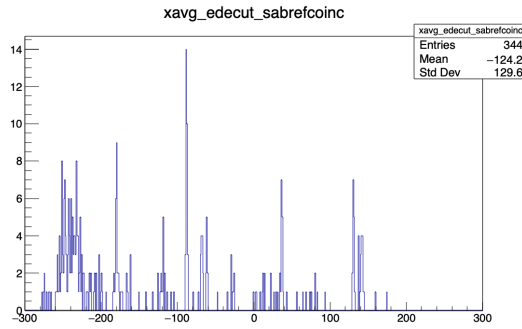
(b) xavg gated on αs and SABRE coincidences

Figure 7: xavg plots with no shifts and wide open windows

is evident that even with bad windows and no shifts, for simple data like this the event builder does reasonably well. Our states are there and sharp, but there are clearly fake coincidences being built with SABRE. We should be able to cut down on these.

So now lets update our `analyzer_input.txt` to reflect our shifts as shown:

```
1  Ztarget: 6 Atarget: 12 Zproj: 2 Aproj: 3 Zeject: 2 Aeject: 4
2  BeamE(MeV): 24 Angle(deg): 20 Bfield(G): 9500
3  Board_shift_file: ./etc/ShiftMap_April2020.txt
4  Scint_offset(ps): 0.65e6
5  coincidence_window(ps): 3e6
6  si_fast_coincidence_window(ps): 3.0e6
7  ion_chamber_fast_coincidence_window(ps): 3.0e6
8  DataDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/raw_root/
9  MinRun: -99 MaxRun: 99
10 TimeshiftedDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
       shifted/
11 SortedDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example//sorted/
12 FastDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/fast/
13 AnalyzedDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/analyzed/
```

```
14  SABREChannelMapFile: ./etc/ChannelMap_March2020.dat
15  GainMatchingFile: ./etc/March2020_gainmatch_2.0V_5486Am241.txt
```

We've added our scintillator offset and I've now included a board offset file for SABRE, which I've made from other portions of this data. Now lets change `process` to

```
1  #!/bin/zsh
2
3  #./bin/binary2root binary2root_input.txt
4  ./bin/analyzer -s analyzer_input.txt
5  ./bin/cleaner cleaner_input.txt
```

This makes it so that we don't run the binary converter again (we only need that once) and we run the `cleaner` after the `analyzer`. We're still running the slow option, since we don't know what our window sizes should be yet. Now open up the histogram file and your `anodeBackTime_toScint` histogram and you should see Fig. 8. Now lets also look at
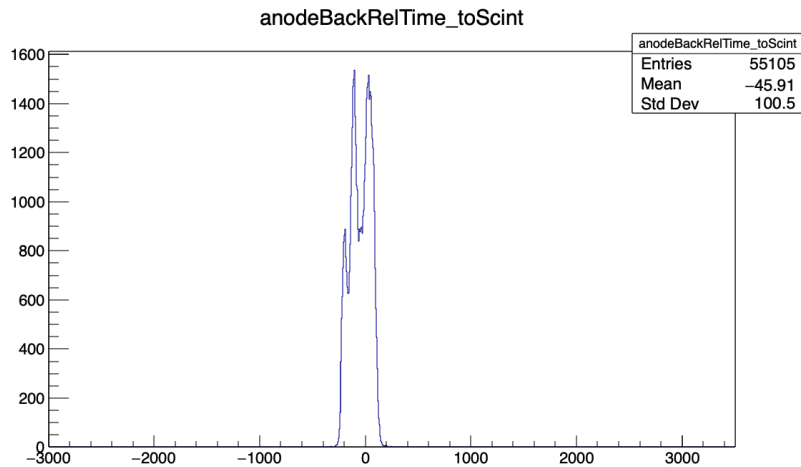

Figure 8: Anode time relative to scintillator time with a shifted scintillator

a slightly different histogram: `delayRelFrontTime_toScint`. This is a histogram of the maximum delay time for the front wire relative to the scintillator, shown in Fig. /refdelayRel. This shows us how wide our "slow" coincidence window should be! The delay lines are by far the slowest portion of the data set, and since everyone else is centered at the same time stamp, we only next to extend far enough to catch the maximum delay. NOTE: you also need to account for the width of the centered peaks (i.e. the width fo the anode relative to scint peaks) in this window as well. Typically this is just adding a small extension onto the window to the delay time. For this data set 1.5 μs is good for a slow window. For our fast window, we can use the width of the anode relative to the scintillator peak to make our condition. This condition basically says that if the anode relative time doesn't fall within this window, it is not a good focal plane event! We can do the same
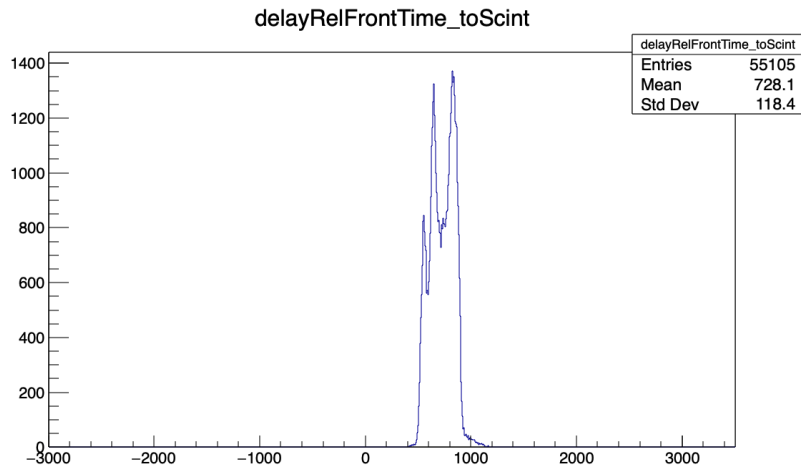
Figure 9: Maximum front delay timestamp minus scintillator timestamp

thing for the SABRE relative to the scintillator time, and I'll include this without showing the plots for this data since there isn't really much to see with a stable residual nucleus.

Ok now lets make one last update to our `analyzer_input.txt` as shown:

```
1  Ztarget: 6 Atarget: 12 Zproj: 2 Aproj: 3 Zeject: 2 Aeject: 4
2  BeamE(MeV): 24 Angle(deg): 20 Bfield(G): 9500
3  Board_shift_file: ./etc/ShiftMap_April2020.txt
4  Scint_offset(ps): 0.65e6
5  coincidence_window(ps): 1.5e6
6  si_fast_coincidence_window(ps): 0.125e6
7  ion_chamber_fast_coincidence_window(ps): 0.25e6
8  DataDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/raw_root/
9  MinRun: -99 MaxRun: 99
10 TimeshiftedDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/
        shifted/
11 SortedDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example//sorted/
12 FastDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/fast/
13 AnalyzedDir: /Users/gordonmccann/Desktop/GWM_EventBuilder/example/analyzed/
14 SABREChannelMapFile: ./etc/ChannelMap_March2020.dat
15 GainMatchingFile: ./etc/March2020_gainmatch_2.0V_5486Am241.txt
```

You can see that we've now added our tighter windows in. I've included SABRE fast windows from other portions of the data set. Now lets change `process` so that we have the full fast sorting on as shown:
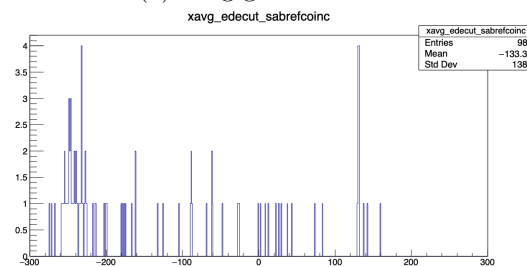
```
1  #!/bin/zsh
2
3  #./bin/binary2root binary2root_input.txt
4  ./bin/analyzer analyzer_input.txt
5  ./bin/cleaner cleaner_input.txt
```

Here all we've done is remove the `-s` so that it uses fast sorting as well. Now lets run it one last time and see what we get. If you open up your histogram file and look at the same xavg plots from before you should see something like Fig 10.
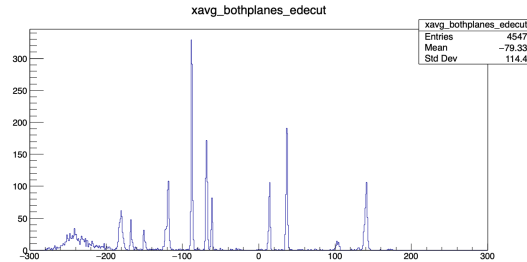
(a) xavg gated on αs

(b) xavg gated on αs and SABRE coincidences

Figure 10: Histograms of xavg with fast coincidence and good windows

You can see that while the xavg plot only gated on αs hasn't changed much, but our coincidence plot has become mostly empty. This is good! With a stable residual, we don't expect really any SABRE coincidences. Obviously with better gates and windows and shifts this could be reduced even further, but for the purposes of this example it demonstrates the goal and functionality of the event builder. Remember, the fast sorting is where each individual experiment will require unique conditions. For this data the anode/scintillator relationship provided a good definition of a focal plane event, and the scintillator/SABRE relationship provided a good definition of a SABRE event. This will not be the case for every experiment.

6   Closing remarks

This code is still under constant development, and any feedback/changes/bugs can be reported to the github remote repository where these will be taken into account. Additionally, if you have quesitons, feel free to email me at gmccann@fsu.edu.

Good luck!