Stephens page of things

Found below is a list of code written by me (Stephen) to setup TIGRESS and GRIFFIN with the minimal amount of effort.

Contents

Maintaining Compatibility

DAQ Setup and Calibration Files
- PSC Tables and Configuration Files
  - Adding Experiments
- Custom Channel Setup in ODB
  - Pole Correction Parameters
  - DAQ/hosts/collector0xN Pages
  - PSC Calibration Parameters
  - Adding Options
- Calibration File Constructors
- Missing Channels

Germanium Calibrations
- Online (Beam) Calibration
- TIGRESS Segment Calibration

Alpha Calibration

LaBr3 Calibration

Efficiency Curve Codes

Other Useful Things
- Resolution Checking
- Unpacking Midas Files

Maintaining Compatibility

All the codes found below will work for both TIGRESS and GRIFFIN, please try and keep it that way. The main thing to look out for when maintaining compatibility is in the definition of GetArrayNumber() in GRSISort. For TIGRESS the 1st array position is 0 whereas in GRIFFIN the first is 1, this can cause problems when defining array sizes.

DAQ Setup and Calibration Files

PSC Tables and Configuration Files

Updated Jun 2020. All experimental setups defined in functions.h header file . For the TIGRESS and GRIFFIN DAQs an PSC table must be constructed to tell which digitizer what detectors are connected to it. I have created a script Conf_Setup.cxx (which requires the header files 'CalibrationParameters.h' and 'functions.h') found here (http://docodon.triumf.ca/~sgillespie/DAQ/), which will can be used to easily construct PSC tables assuming the detectors are cabled in a standard manner (as defined here for GRIFFIN and TIGRESS). The header file 'CalibrationParameters.h' contains all the calibration parameters which for Germanium, LaBr3 and Silicon (not PACES you can make your own script) are generated by the calibration scripts below. If no calibration parameters are given the gains and offsets are set to 1 and 0 (uncalibrated). The header file 'functions.h' contains the cabling of individual detectors (TIGRESS/GRIFFIN/TIP etc.) The script is compiled with the command:

```
g++ Conf_Setup.cxx -std=c++0x -o SetupConfFile
```

This script can take up to three arguments, but only one is required for the script for it to work. The script will create a configuration file 'Default name Conf_File.txt' which is used by several of the script found here so this script should always be run first. The compiled script is run with:

```
./SetupConfFile exp-name PSCTable.txt(optional) SegmentParameter.txt(optional)
```

Where exp-name is the name of the specific experimental setup (tigress/griffin etc.). The script will work for all current experimental setups, but as I lack the foresight to think of future experimental setups someone will have to edit the script in the future. Currently supported experiment types are:

Setup Name	Description
tigress	TIGRESS 16 Clover Configuration
emma	TIGRESS 12 Clovers + SSB + EMMA Focal Plane
emmas3	TIGRESS 12 Clovers + Upstream S3 + SSB + EMMA Focal Plane
griffin	GRIFFIN 16 Clover + PACES + SCEPTAR + ZDS + LaBr3
bambino	TIGRESS 16 Clover + 2 S3 Detectors Standard Arrangment
descant	GRIFFIN 16 Clover + PACES + SCEPTAR + ZDS + LaBr3 + DESCANT
tip	TIGRESS 16 Clover + TIP
Any other setup	See here.

The SegmentParameter.txt parameter is only needed for TIGRESS setups and contains the segment calibration parameters (if you actually need them calibrated). PSC tables are only constructed if a second argument is provided when running the scripts. PSC tables are loaded into the DAQ via the odbedit program which is a installed on the DAQ accounts. To update the PSC table do the following

1. Log into the DAQ computer via

```
ssh -X griffin@grsmid00 / tigdaq@grsmid02
```

2. Run the command

```
odbedit -c @PSCTable.txt
```

3. Enjoy your new PSC Table.

It should be noted that the PSCTable does not actually include the calibration parameters from 'CalibrationParameters.h'. It could be changed to do that but I decided against it. To add calibration parameters to the PSC table use the code CustomChannelSetup.cxx described below. All of the above assumes that everything is cabled in a standard manner, but (preempting Adam's inevitable comment) I have given the script the capability to accept non standard cabling. In the event that a detector channel is not in standard channel add an entry to a file custom.dat with the format:

```
MNEMONIC CollectorNumber PortNumber ChannelNumber
```

Where MNEMONIC is the detector name as defined by the standard detector nomenclature found here. CollectorNumber, PortNumber and ChannelNumber refer to the GRIF-C, GRIF-16 and channel number the detector is connected to. **Note:** The CollectorNumber, PortNumber and ChannelNumber must be in hexadecimal format and start counting from 0, so the first channel in an GRIF-16 is 0.

Adding Experiments

At present the script will generate the current TIGRESS and GRIFFIN experimental setups, but it can be easily edited to do others. To add another experiments you need to edit the following in Conf_Setup.cxx:

1. int num_known_exp. Seems self evident increase the number as more things are added.
2. string exp_names[]. Add an identifier to call a specific experimental setup.
3. string exp_description[]. Add a description of what the experimental setup is.
4. switch(option). This is where the new functionality to be added. The option should call functions from the header file 'functions.h' for any combination of detectors for example

```
makeTIGRESS(arguments..)
makeS3(arguments..)
```

5. Add any new functions to the file 'functions.h', use other examples as a guide.

Please try to keep these as general setups assuming standard cabling. There is already functionality to do non standard cabling.

Custom Channel Setup in ODB

I have created a general purpose script CustomChannelSetup.cxx which can be used to automate some of the more dull

tasks required when setting up the DAQ. Five header files 'tigadc.h', 'grifadc.h', 'tigPZ.h', 'grifPZ.h' and 'CalibrationParameters.h' are required to run the script all of which are also found here (http://docodon.triumf.ca/~sgillespie/DAQ/). The script is compiled with the command:

```
g++ CustomChannelSetup.cxx -std=c++0x -o CustomChan
```

The compiled script takes up to 3 arguments of which two are required, it is run with the command:

```
./CustomChan option experiment outfilename(optional)
```

Where option is the specific task to perform, there are currently 3 options defined which are detailed below. Experiment is tigress/griffin and as I couldn't be bothered to make it case insensitive they must be lower case. The output of the script is a text file outfilename (Default: CustomChan.com) which contains a bunch of commands to be run via odbedit. To run these commands do the following.

1. Log into the DAQ computer via

   ```
   ssh -X griffin@grsmid00 / tigdaq@grsmid02
   ```

2. Run the command

   ```
   odbedit -c @CustomChan.com
   ```

Pole Correction Parameters

Perhaps the most common and annoying task is setting the pole correction parameters for the Ge detectors whenever detectors are moved within the array. The pole correction values for GRIFFIN and TIGRESS are contained in 'grifPZ.h' and 'tigPZ.h' with the values taken here for GRIFFIN and here for TIGRESS. An additional file 'gePositions.txt' is needed to run the script. This file is simply a 16 line text file where each line corresponds to an array position. The file will need to be edited with the current array configuration e.g If TIG07 is in position 1 the 1st line should read 7. If an array position is empty mark it as 0. To generate the command script for setting pole correction values use.

```
./CustomChan pz tigress/griffin outfilename(optional)
```

and run the output with odbedit as described above.

DAQ/hosts/collector0xN Pages

In the event that the grif16 arrangement is radically changed the script is capable of filling in the collector0xN arrays needed to get the DAQ to work. The current grif16 arrangement for the TIGRESS and GRIFFIN DAQs is contained in 'tigadc.h' and 'grifadc.h' which should be updated as needed. Run this command as:

```
./CustomChan adc tigress/griffin outfilename(optional)
```

and run the output with odbedit as described above.

PSC Calibration Parameters

Another task that I rarely bother with for TIGRESS, but is common amongst GRIFFIN is setting the gain and offset (and soon to be) quadratic parameters in the PSC table. To do this a configuration file 'Conf_File.txt' and the header file 'CalibrationParameters.h' are required. The configuration file 'Conf_File.txt' can be generated with the code SetupConfFile, described in the previous section 'PSC Tables and Configuration Files'. The calibration parameters are contained within arrays in 'CalibrationParameters.h' and are generated by the calibration scripts described below. Run the command:

```
./CustomChan gain tigress/griffin outfilename(optional)
```

and run the output with odbedit as described above.

Adding Options

At present the script only does 3 things, but it can be easily edited to do more using the other options as guides. To add another option you need to edit the following:

1. int num_options. Seems self evident increase the number as more things are added.
2. string custom_options[]. Add an identifier to call the specific function.
3. string option_description[]. Add a description of what the new option does.
4. switch(option). This is where the new functionality to be added. The option should generate a script containing commands which are used by odbedit to update the odb page.

Calibration File Constructors

Using the Configuration File created from Conf_File.cxx calibration files for GRSISort can be constructed with the script 'CalFileConstructor.C' found here (http://docodon.triumf.ca/~sgillespie/DAQ/) The script is fairly basic and simply formats the configuration file into the format required by GRSISort. The script is compiled with the command

```
g++ CalFileConstructor.C -std=c++0x -o ConstructCalibrationFile
```

The script can take up to 3 arguments all of which are optional. To run the compiled script use the command:

```
 ./ConstructCalibrationFile ConfigurationFileName(optional) CalibrationFileName(optional)
EMMAConfigurationFileName(optional)
```

If no arguments are provided it defaults to ConfigurationFileName = 'Conf_File.txt' and CalibrationFileName =

'CalibrationFile.cal'. The EMMA Focal plane uses a separate DAQ system and as I can't make them keep a standard setup the EMMA configuration file it has to be defined manually. An example of the Configuration File Format 'Conf_File_Emma.txt' is found in DAQ Codes (http://docodon.triumf.ca/~sgillespie/DAQ/)

Missing Channels

To check if specific channels are missing in the DAQ I have created the script 'chan_check.cxx'. The script require two header files 'tigadc.h' and 'grifadc.h' all of which are found here (http://docodon.triumf.ca/~sgillespie/DAQ/) The script requires GRSISort to be installed and is used to determine if a channel is missing and to output to a text file the name of the missing channel as well as the grifadc number and channel number it should be connected to. To compile the script run:

```
g++ chan_check.cxx -std=c++0x -I$GRSISYS/include -L$GRSISYS/libraries `grsi-config --cflags --all-
libs` `root-config --cflags --libs` -lTreePlayer -lSpectrum -o ChanCheck
```

The compiled script can take up to 4 arguments of which two are required. To run this script use:

```
./ConstructCalibrationFile FragmentTreeFilePath ConfigurationFileName CalibrationFileName(optional)
OutputHistFileName(optional)
```

Where FragmentTreeFilePath is the name of the Fragment Tree containing the data, ConfigurationFileName is the Configuration File for Conf_file.cxx. CalibrationFileName and OutputHistFileName default to CalibrationFile.cal and channels.root if arguments are not provided. When this script is run it will produce a text file 'missing.txt' containing the missing channel information and is format as

```
MNEMONIC grifadc-number grif16-channel-number
```

The scripts will take the grifadc-number from the header files 'tigadc.h' and 'grifadc.h' which contain the current grif16 arrangement in the two DAQs so need to be kept up to date. The script will determine from the Configuration File if the data from the TIGRESS or GRIFFIN DAQ.

Germanium Calibrations

Updated Jun 2020. All calibration histograms are now made by make_ge_histograms.C .

Germanium Calibrations Codes are found here (http://docodon.triumf.ca/~sgillespie/Calibration/).

There are 4 scripts which perform the calibration, calib-ge.cxx, make_ge_histograms.C, linear_energy.C and quad_energy.C as well as one script which fixes bad calibration, calib_fix.C. A README file is also included which explains how the script works. The scripts will work for both TIGRESS and GRIFFIN without any modification and require GRSISort to be installed.

To perform a calibration follow the below steps:

Note: It is essential that one of the sources used is 60Co as the linear energy script expects this and will fail otherwise.

1. Copy all of the scripts into a directory and insert a calibration file with the name CalibrationFile.cal. This calibration file is needed for channel mapping purposes only.
2. Compile the script calib-ge.cxx with the command

```
g++ calib-ge.cxx -std=c++0x  -o Calibrate
```

3. The calibration is then performed using

```
./Calibrate SourceName1 AnalysisTreeFilePath1 ... SourceNameN AnalysisTreeFilePathN
```

   Where SourceNameN is the name of the sources used and AnalysisTreeFilePathN is the location of the analysis tree containing that data. The source name does not have to be in an exact format and multiple different formats will work (60Co, 60co, Co60, co60). If you think there is a common source format not included here, you are wrong and should feel bad.

4. After the calibration is complete a file 'quad_energy_coeff.txt' will be produced which contains the calibration parameters in three arrays. These arrays should be copied to 'CalibrationParameters.h' to construct the calibration files or to set the parameters in the PSC table. Remember to recompile and re-run the SetupConfFile code (here) before constructing a calibration file, or the changes to the header file will not stick.
5. Fitted histograms, fit function and residual plots are saved to the file 'quad_cal.root', and can help you to judge the quality of the energy calibration.

Online (Beam) Calibration

The calibration scripts can also be used to perform an online calibration, however this requires some modification:

Note: The script linear_energy.C must still be run for a 60Co source and have produced the file 'lin_energy_coeff.txt' before an online calibration can be performed.

1. In quad_energy.C change the energies in the arrays source_energy[4] and source_energy_er[4] (lines 252 and 259) to the online calibration energies. The 5th element of the array num_peaks[] must also be changed depending upon how many peaks are used for the calibration.
2. Run the command

```
./Calibrate SourceName1 AnalysisTreeFilePath1 ... SourceNameN AnalysisTreeFilePathN
```

   where SourceNameN is replaced by Online/online.

TIGRESS Segment Calibration

To calibrate the TIGRESS segments use the script 'seg_cal.cxx' found here (http://docodon.triumf.ca/~sgillespie/Calibration/).

The script works by details to be added later

Alpha Calibration

Will go here at some point.

Instructions for calibrating S3 detectors are here.

LaBr3 Calibration

Found here (http://docodon.triumf.ca/~sgillespie/LaBr/)

Basically identical to the Ge codes

Efficiency Curve Codes

Ge efficiency code is found here (http://docodon.triumf.ca/~sgillespie/EfficiencyCurveCode/GeEff). Instructions for running are in the README file

LaBr still needs to be done.

Other Useful Things

Resolution Checking

To perform a quick check of detector resolutions using a 60Co source I have a script 'res_check.cxx' found here (http://docodon.triumf.ca/~sgillespie/Misc/). The script sorts data into histograms, fits the peaks and outputs to both a text file and the terminal the detector array positions and resolutions, with particularly bad resolutions output to the terminal in red. **Note:** The definition of array position is different in GRSISort for TIGRESS and GRIFFIN, with the first array position being 0 for TIGRESS and 1 for GRIFFIN. For the sake of array sizes the histograms use the TIGRESS definition of Array Number, but the resolutions output to the terminal use the GRSISort definitions. Compile the script with the command:

```
g++ res_check.cxx -std=c++0x -I$GRSISYS/include -L$GRSISYS/lib `grsi-config --cflags --all-libs
--GRSIData-libs` -I$GRSISYS/GRSIData/include -L$GRSISYS/GRSIData/lib `root-config --cflags --libs`
-lTreePlayer -lMathMore -lSpectrum -lMinuit -lPyROOT -o ResCheck
```

The script takes up to three arguments of which only one is required, run the script using:

```
./ResCheck AnalysisTreeFilePath CalibrationFileName(Optional) OutputHistFileName(optional)
```

Where AnalysisTreeFilePath is the analysis tree containing the 60Co source data, CalibrationFileName is the cal file (Default:CalibrationFile.cal) and OutputHistFileName contains the sorted and fitted histograms. This script was written for GRSISort version 4.0, if you are using an older version of GRSISort, stop doing that and update it.

Unpacking Midas Files

To quickly unpack Midas files into Fragment and Analysis Trees I have a bash script 'MakeAnalysis.sh' found here (http://docodon.triumf.ca/~sgillespie/Misc/) which I use during experiments. The file paths defined in the script will need to be edited then the script can be run with the command

```
./MakeAnalysis.sh RunNumber(optional)
```

If given no arguments the script will unpack all the Midas files in a directory (after a certain run number defined on line 8) and moves the created Analysis Trees and Fragment Trees into different directories. The script will not try to unpack an Analysis Tree which already exists unless the Midas file is newer than than Analysis Tree (this is a useful feature during experiments). The script can take a run number as an optional argument in which case it will only unpack the Midas file of a single run (and its subruns).
