

Timing analysis framework

LGADUtils

V. Gkougkousis

Institut de Física d'Altes Energies - CERN

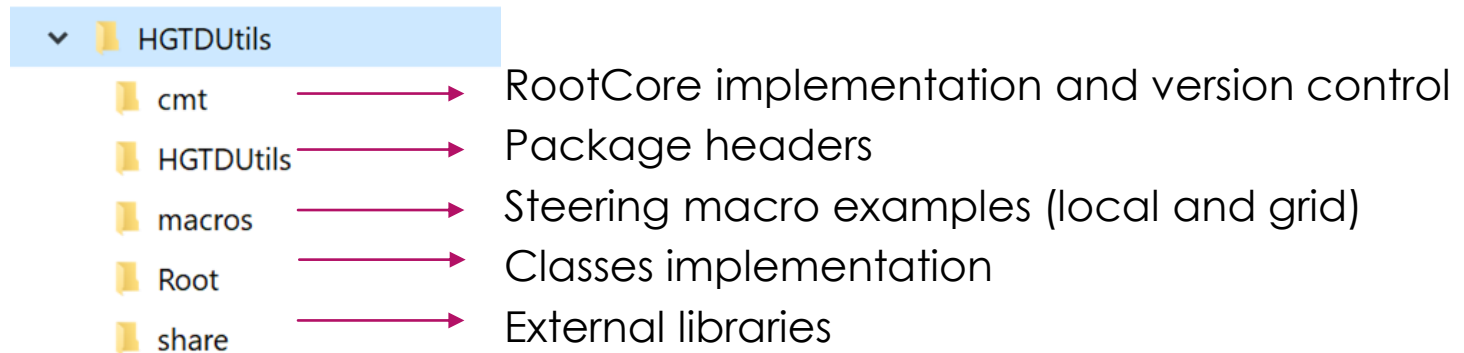
CERN – 15 / 10 / 2020

•Timing analysis Framework

C++ 11

Why does it makes sense?

- Code available on git: <https://gitlab.cern.ch/egkougko/lgadutils>
- Structure:



- Following standard ATLAS analysis package organization
- Running as interpreted (CINT), compiled (CMake) or RootCore version
- Tested in multiplatform environment (Windows, Linux), appropriate preprocessor instructions incorporated for compatibility
- Only requirement ROOT (with **FFTW** and **RooFit**), no ATLAS software
- Validated with ROOT 5 and production version of ROOT 6
- Allows users to control their analysis by creating a top level steering macro
- Combines all steps, from data conversion starting from RAW or txt files to plotting

•Timing analysis Framework

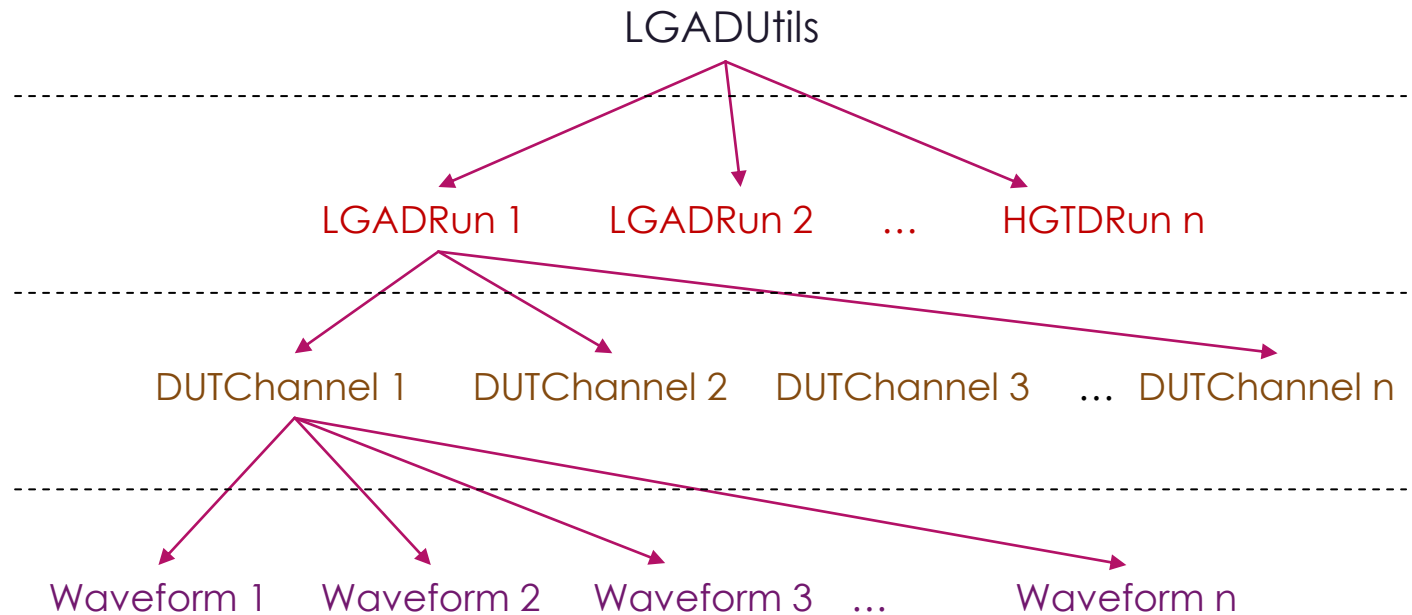
C++ 11

How is the logic structured?

- Four main classes with dedicated header and implementation files, one wrapper class handling user interaction

- **LGADBase** → Wrapper to handle user I/O and pass arguments
- **LGADUtils** → Basic framework function and infrastructure
- **LGADRun** → Timing resolution, CFD maps, multi DUT operations
- **LGADChannel** → Mean pulse shape, mean pulse properties form entire run
- **WaveForm** → Single Waveform properties and time walk corrections
- **Bonus: LGADSel** → **Selector Class with auto-set 64 channel support**

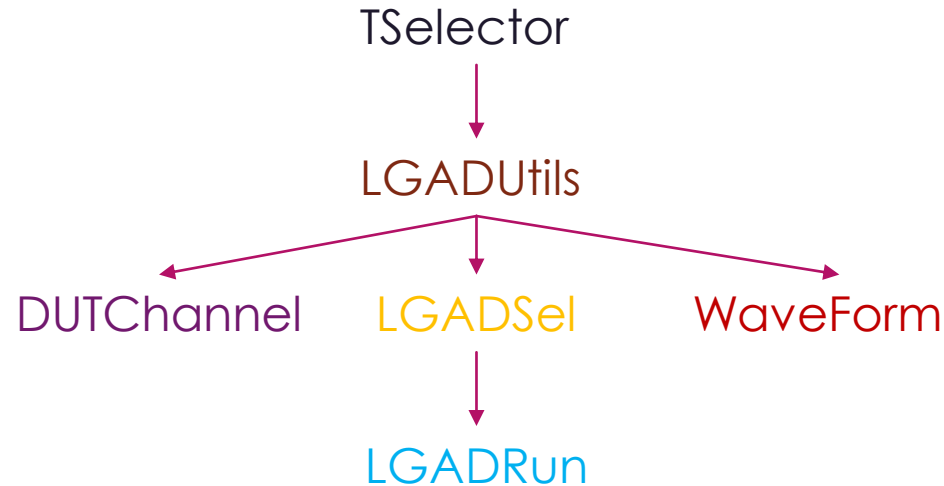
1st level
2nd level
3rd level
4th level



•Timing resolution Framework

Inheritance

C++ 11

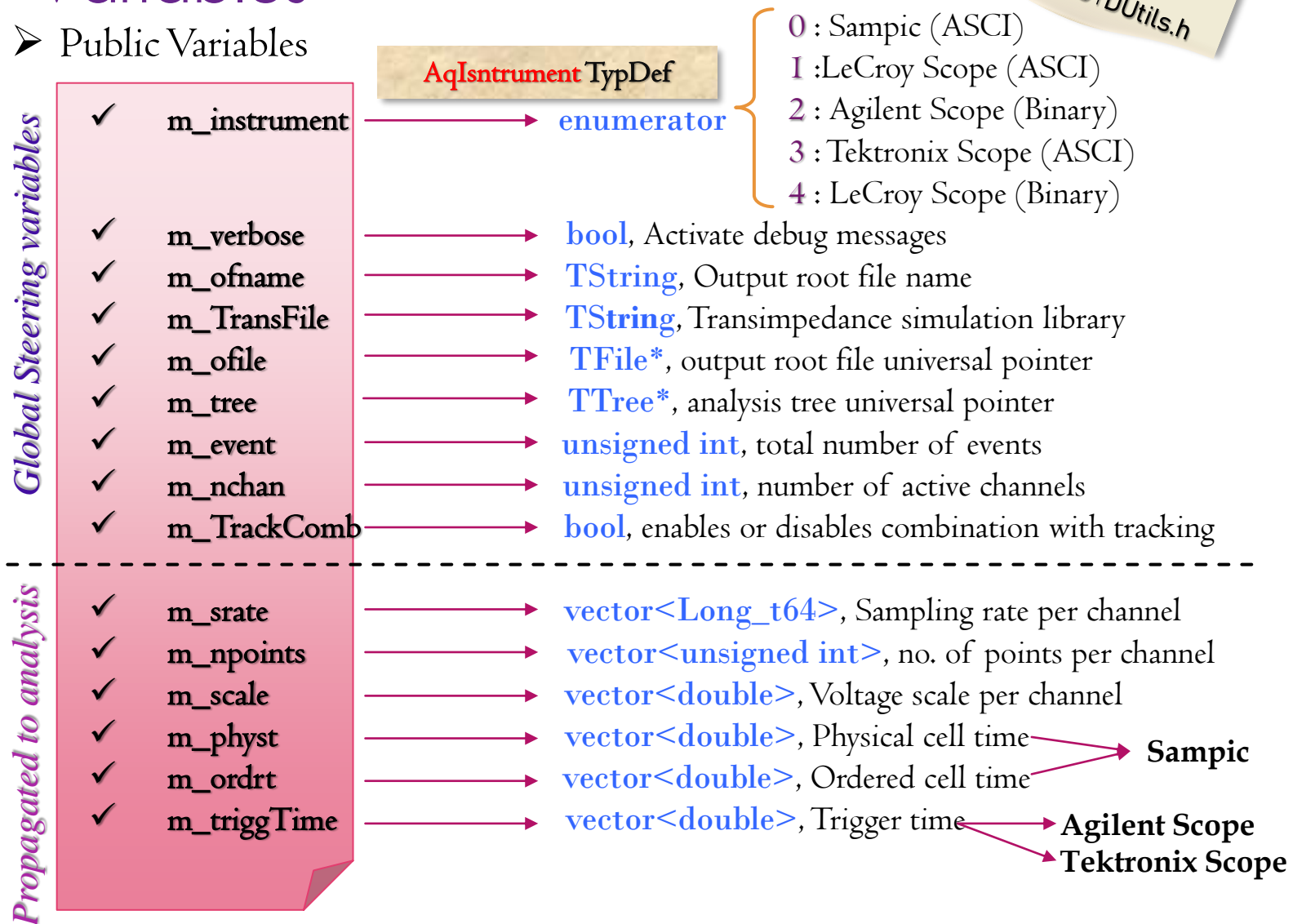


- Inheritance from a `TSelector` allows for seamless analysis with internal or external root files
- Normal selector methods implemented in `LGADRun` class
- `LGADSel` is a wrapper for setting the pointers, branch address, figuring out number of active channels and used instruments and doing the necessary mapping
- Universal access to the `LGADUtils` helper and fitting infrastructure

•LGADUtils Class

Variables

➤ Public Variables



•HGTDUtils Class

File:
HGTDUtils.cxx

Public functions

➤ Constructor and Destructor

LGADUtils()

virtual ~LGADUtils()

Empty constructor, clear all vectors and write single 0, set default instrument to LeCroy and all variables to 0

➤ Methods to set global variables

SetInstrument(AqInstrument)

Set the instrument from supported types

bool SetSRate(**double** rate, **unsigned int** ch)

Set sampling rate per channel. If not set by user, it will automatically be set during data conversion

bool SetNPoints(**unsigned int** points, **unsigned int** ch)

Set n. of points rate channel. If not set by user, it will automatically be set during data conversion or nTpule analysis

SetInDataNames (**TString** DataDir , **TString** DataName. **TString** ext)

Set data directory, name of files and extension. If extension left black, it is automatically managed by instrument. (default is empty)

SetOutDataNames (**TString** DataDir, **TString** DataName)

Set output directory and filename. If left blank uses input directory and data name without extension (default is empty)

SetTrackInDataNames (**TString** DataDir, **TString** DataName)

Set Tracking directory and filename to include in the analysis (default is empty)

•LGADUtils Class

File:
HGTDUtils.cxx

Public functions

➤ Methods to get global variables

- AqInstrument** GetInstrument() → Returns the instrument set (integer value)
double GetSRate(**unsigned int** ch) → Returns sampling rate for specified channel
unsigned int GetNPoints(**unsigned int** ch) → Returns No. of points for selected channel
bool GetTrackComb() → Returns True if tracking is included and false otherwise

➤ General Helper functions

- string** reduce(**const string** str, → Replaces or removes characters in a string
const string fill, with the specified sequence in “fill”.
const string whitespace) Default is empty.
string trim(**const string** str, → Removes characters or spaces from the
const string whitespace) beginning and end of the string
unsigned int CountFiles(**const char*** dir, → Returns number of files of extension
const char* ext) “ext” in folder “dir”. POSIX dependent.
int DirExists(**const char*** path) → Checks if folder exists. Returns 1 if it
does, 0 if not.
-
- template** Derivate(**template** *T, **int** start = 1) → Analytically computes point derivative of
vector w starting at index start.
double Mean(**template** *T, → Compute standard mean value of vector
int start = -1, **int** stop = -1) W from element start to stop
double Stdev(**template** *T, → Compute standard deviation of vector w
int start = -1, **int** stop = -1) from element start to stop

Supported
types

int
bool
double
float

•LGADUtils Class – Data Converters

Data Converters

File :
DataConverters.cxx

Tektronix® TDS series



Agilent
Technologies

Infiniium Series

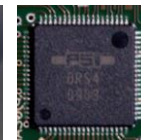
Dedicated Converters for
multiple instruments with up
to 64 native or daisy
chained channels



Sampic Series v2.0
(16, 34, 68 ch versions)



DRS4 (natively)



LeCroy WaveRunner



- Completely transparent for the user, only need to set the **m_instrument** correctly and appropriate converter will be called
- Input/output directories have to be set, extension if not set are automatically assigned
- Converter will figure out number of channels and will create output Ntuple with respect to instrument and channels

•LGADUtils Class – Data Converters

Data Converters

File :
DataConverters.cxx

➤ Dedicated Functions

bool ConvertData()

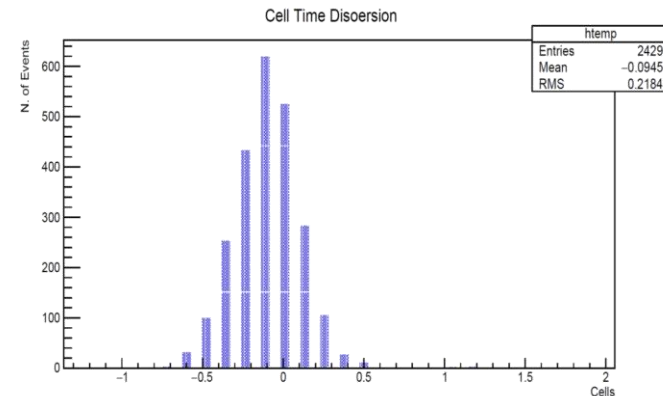
Starts the data conversion with the parameters set. m_instrument and input/output dirs. Must be already defined

void SetStartStopEvt(**unsigned int** Evntt1,
unsigned int Evnt2)

Set the event to start the conversion from and where to stop. If not called, it goes through the entire run. Effective only in Tektronix and LeCroy readout mode.

➤ Time Alignment

- *Effect visible on Sampic and when more than one oscilloscope are daisy chained*
- *All channels should trigger simultaneously*
- *Trigger time difference between channels presents a binned Gaussian peaked at 0*
- *Bin size is equal to sampling rate*



Correction

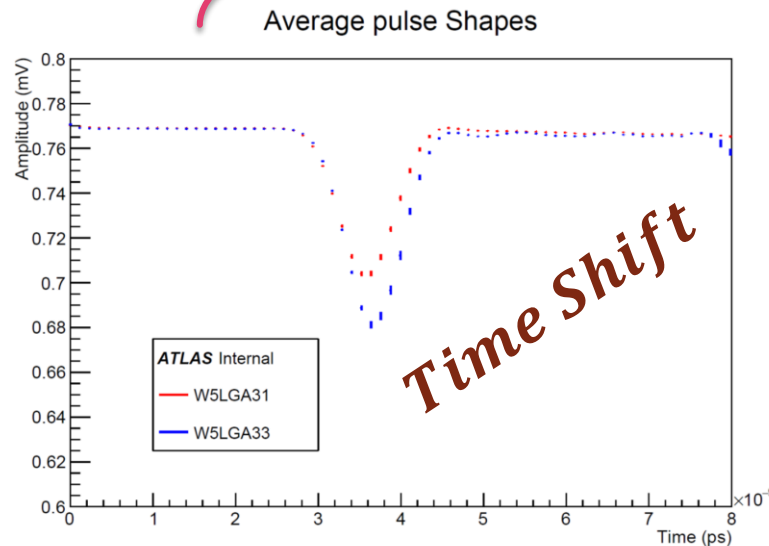
Time for each channel needs to be recalibrated with respect to a channel of reference

•LGADUtils Class – Data Converters

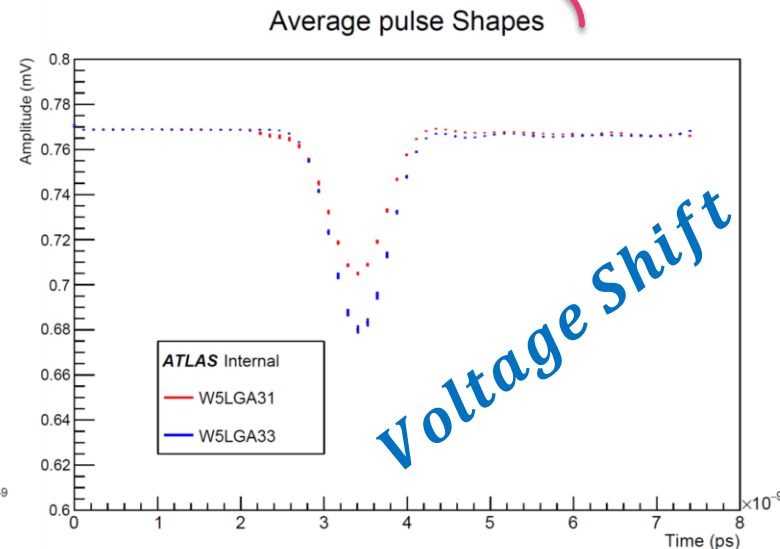
Time Alignment

Two possible correction methods

File :
DataConverters.cxx



- Assume that reference channel start always at 0
- Adjust the start time of additional channels to compensate trigger time mismatch



- Assume all channels start at 0
- Shift voltage values to any of the channels to the end of the waveform to compensate for trigger mismatch
- Works for small mismatches

•LGADBase Class – Data Converters

Root Tree Branches per instrument

- Universal Branches

tXX (std::vector double)	: Per channel vector of time values
wXX (std::vector double)	: Per channel vector of amplitude values
EvntNo (UInt_t)	: Event number assigned at conversion

- Sampic Branches

phystXX (std::vector double)	: Per channel vector of ordered time
ordrtXX (std::vector double)	: Per channel vector of physical time
nPoints (UInt_t)	: No. of recorded points for all channels
SnRate (Long64_t)	: General Sampling rate for all channels

- Scope Branches

vScaleXX (float)	: Per channel vertical scale
nPointsXX (UInt_t)	: Per channel no. of recorded points
SnRateXX (Long64_t)	: Per channel sampling rate

- Tektronix/Agilent ScopeXX (UInt_t) : No. of Physical scope that channel belongs to
- Binary Branches
- Tektronix TXT Branches

triggTimeXX (double)	: Per channel trigger time
-----------------------------	----------------------------
- Labview/Tektro-nix Binary Branches

triggtime (double)	: Universal event trigger time for all channels
---------------------------	---

•LGADUtils Class - HGTD Fits

File: HGTD Fits.cxx

RooFit Implementation

- int** GaussFit(**vector**<**double**> *w, —————→ Performs Gaussian fit on vector w, returns mean and sigma with uncertainties. Points sets the start and stop index of fitting
 pair<**double**, **double**> &gmean,
 pair<**double**, **double**> &gsigma,
 pair<**int**, **int**> points)
- int** LinearFit(**vector**<**double**>* vec, —————→ Performs Linear fit on w, returns slope and intercept with errors. Fit limits adjusted automatically.
 pair<**double**, **double**> &slope,
 pair<**double**, **double**> &intersept)
- int** GauLinear(**std::vector**<**double**>* vec, —————→ Performs Gauss X Linear fit on w, returns mean and sigma with uncertainties.
 pair<**double**, **double**> &magMPV,
 pair<**double**, **double**> &magSigma)
- int** GauLandauFit(**vector**<**double**>* vec, —————→ Performs Gauss X Landau fit on w, returns MPV and sigma with uncertainties.
 pair<**double**, **double**> &magMPV,
 pair<**double**, **double**> &magSigma)
- float** LinearInter(**float** xI, **float** yI, —————→ Tow point linear interpolation on y3, between (xI, yI) and (x2 y2).
 float x2, **float** y2, **float** y3)
- double** FFT(**vector**<**double**> *w, **int** snrate, —————→ Fast Furrier transform on w, from point start to stop. It returns the first order frequency.
 int start, **int** stop)

Limits for all fits are automatically adjusted with respect to std & mean of w
Return codes are indicative of failure type (Minuit fit quality, vector elements...)
Implemented quality requirement (Minuit Covariant has to succeed)

•LGADSelector Class

Features

File: HGTDSEL.cxx

- Holds all required pointers and branches for all instruments and cases
- Dynamic definition of number of active channels from nTuple branches, no upper limit to channel number
- Will automatically determined the number of channels saved in a root file and set appropriate pointers
- Will automatically detect type of instrument from the saved branches in root file and will set m_instrument variable. If in disagreement with what eternally set, Selector prevails
- Will connect the appropriate branches per instrument type
- Resets the vectors and pointers already used in the converters section (if any) and assigns them to nTupule branches
- Shell class for the analysis, written in the HGTDRun class

•LGADSelector Class

File: HGTDSEL.h

Ntuple Variables

General

- ✓ EvnNo
- ✓ w0I - wXX
- ✓ t0I - tXX

unsigned int, event number

double, channel amplitude in millivolts

double, channel time in picoseconds

Agilent, LeCroy, Tektronix

- ✓ vScale0I - vScaleXX
- ✓ nPoints0I - nPointsXX
- ✓ SnRate0I - SnRateXX

float, voltage scale per channel

unsigned int, number of points per channel

Long_64t, Sampling rate per channel

Agilent

- ✓ triggTimeXX - triggTimeXX

double, trigger time per channel

LeCroy, Tektronix

- ✓ Triggtime

double, global trigger time

SamPic

- ✓ phystXX - phystXX
- ✓ ordrtXX - ordrtXX
- ✓ nPoints
- ✓ SnRate

double, time of the physical cell of the chip

double, time with respect of the general clock

double, global number of points per channel

double, global sampling rate

•Waveform Class

File: WaveForm.cxx

Main Functions

➤ Constructor and Destructor

- `WaveForm()` → Empty constructor
- `WaveForm(vector<double>* voltage, vector<double> time)` → Constructor with voltage and time vectors. It will calculate sampling rate.
- `WaveForm(vector<double>* voltage, int snrate)` → Constructor with voltage vector and sampling rate. I twill generate the time vector.
- `WaveForm(vector<double>* voltage, vector<double> time, int snrate)` → Full constructor with time, voltage and rate. If rate does not agree with time vector, rate is overwritten.
- virtual** `~WaveForm()` → Destructor

➤ Methods to set variables

- void** `SetVoltage(std::vector<double> *volt)` → Set the voltage points vector. This is a pointer
- void** `SetTime(std::vector<double> time)` → Set the time vector
- void** `SetPolarity(polarity pol)` → Define polarity if known. Enumerator typedef
- void** `SetSnRate(int snrate)` → Set the sampling rate
- void** `SetCFDfraction(double fraction)` → Set the CDF fraction for calculations
- void** `SetTrigg(double tri)` → Set the constant threshold trigger
- void** `SetBasePos(basepos pos)` → Define the position with respect to baseline
- void** `SetTransimp(float transimp)` → Set 1st stage amplifier trans-impedence value
- void** `SetAmpGain(float gain)` → Set 2nd stage amplifier gain

•Waveform Class

File: WaveForm.cxx

Analysis roadmap

- A four sequential step analysis approach
- Analysis escalates in a pyramid structure

Five preliminary sequential steps before we even start looking at the waveform

Step 1

Set Waveform values → Determine polarity → Find max, min point → Find start, stop point → Determine if noise → Determine if pulse within window

Step 2

Define noise points → Use Gaussian fit for noise/Pedestal → Pedestal subtraction/inversion → Recalculate start, stop points, min, max

Step 3

Compute charge → Determine rise time → Determine CFD Time → Compute dV/dT → Determine Trigger Time → Estimate Trigger & CFD ToT

Step 4

Perform CFD time to voltage Correction (Time Walk) → Signal FFT → Noise FFT

•Waveform Class

File: WaveForm.cxx

Analysis roadmap

- A four sequential step analysis approach
- Analysis escalates in a pyramid structure

Five preliminary sequential steps before we even start looking at the waveform

Step 1

Set Waveform values → Determine polarity → Find max, min point → Find start, stop point → Determine if noise → See if pulse within window

Step 2

Define noise points → Use Gaussian fit for noise/Pedestal → Pedestal subtraction/inversion → Recalculate start, stop points, min, max

Step 3

Compute charge → Determine rise time → Determine CFD Time → Compute dV/dT → Determine Trigger Time → Estimate Trigger & CFD ToT

Step 4

Perform CFD time to voltage Correction (Time Walk) → Signal FFT → Noise FFT

•Waveform Class

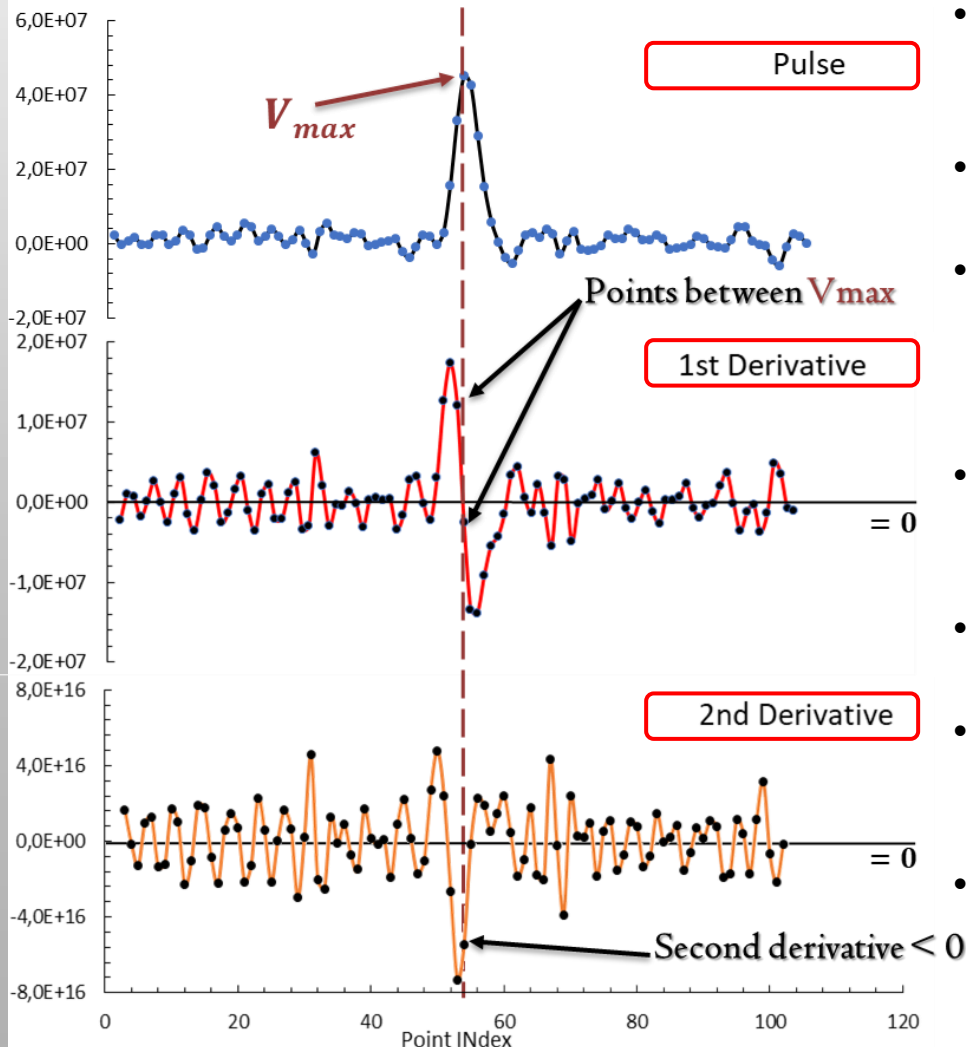
File: WaveForm.cxx

Polarity and Peak detection algorithm

Caution:

No smoothing applied!!

Tow point linear interpolation for precision and to satisfy continuity requirement of local maxima minima theorem



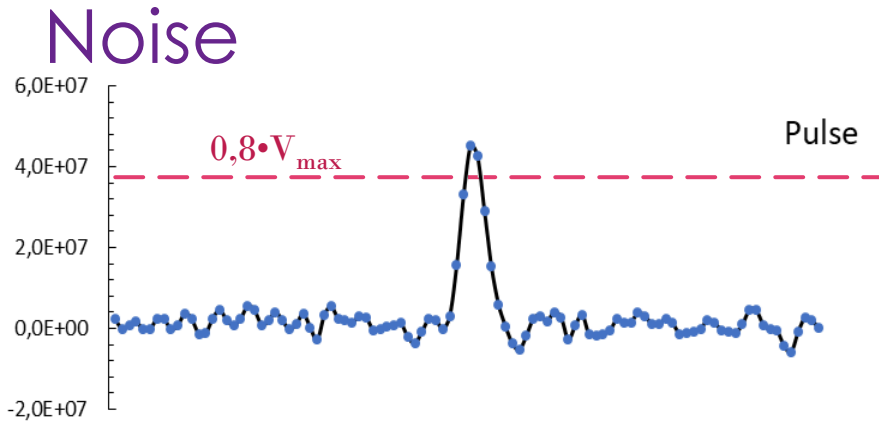
- Calculate analytically the first and second derivative for each waveform
- Find the zero crossing points of first derivative (local extrema)
- Detect sign of the second derivative at each local extrema
 - If $f'' < 0$: local maxima
 - If $f'' > 0$: local minima
- Construct vectors containing the local maxima and local minima amplitude values
- Calculate for each vector the standard deviation and mean value
- Calculate the highest distance from mean value for each vector (D_m)
- If $SD_{maxima} > SD_{minima}$
 $D_m(max) > D_m(min)$
 or $SD_{maxima} < 1.05 * SD_{minima}$
 $D_m(max) > D_m(min)$

Positive polarity

•Waveform Class

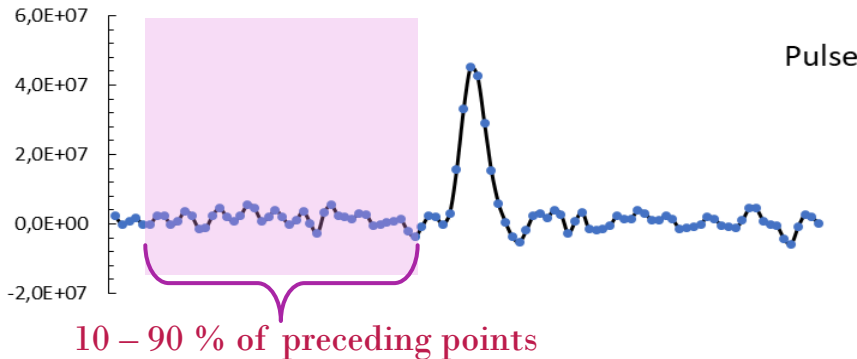
File: WaveForm.cxx

Is Signal?



- Find number of points with value equal to $0.8 \cdot V_{\max}$
- If $N_{\text{points}} > 2$ continue and test 0.7, 0.6 & $0.5 \cdot V_{\max}$ to account for wavy waveforms
- If $N_{\text{points}} > 2$ then require $dN_{\text{points}} < 8/12/16/20$

Noise Points



- Noise calculation include only points before pulse
- Points considered:
 $N_{\text{stsr}} * 10\% \text{ to } N_{\text{stsr}} * 90\%$
- If pulse not within window perform same operation after N_{end}

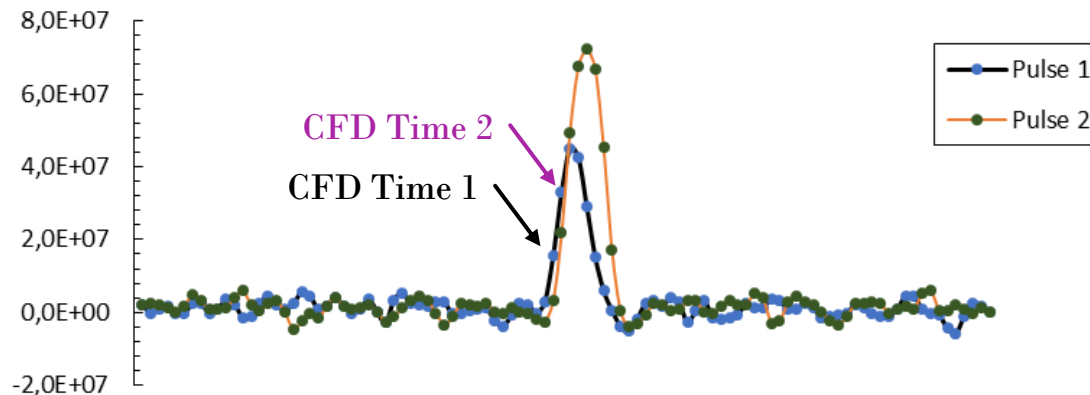
Noise estimation

- Noise defined as σ of Gaussian fit
- Pedestal defined as mean of Gaussian fit
- Gaussian fit limits and range are constrained within ± 5 Standard deviation of amplitude values
- Fit quality check and revert to classic mean and std if minimization fails

•Waveform Class

File: WaveForm.cxx

TimeWalk Correction



CFD Time is different for each waveform, to properly project and sum we need to compensate:

- Time correction
 - Set $t_0 = t_{\text{CFD}}$
 - Readjust time of each point as $t' = t - t_{\text{CFD}}$

Simplest, fastest solution
Non-uniform time bins,
impossible to sum

- Voltage correction
 - Calculate the voltage at t_{CFD} using two point linear interpolation
 - Recalculate the voltage at each time point using two point linear interpolation
 - Readjust the time bins to be at exactly the same point as before

More complex
Uniform time vector
Easy to sum and fit

•Waveform Class

File: WaveForm.cxx

Main Functions

- int** GetPolarity() —————> Returns the polarity of the waveform (pos, neg, unef). It is calculated via a first and second order derivative test using the Fermat theorem of peak detection
- int** GetSnRate() —————> Returns the sampling rate of
- double** GetTrigg() —————> Returns trigger threshold
- float** GetCFDfraction() —————> Returns CFD percentage for calculations
- vector<double>*** GetVoltage() —————> Return original voltage vector
- vector<double>** GetTime() —————> Returns original time vector
- vector<double>** GetAdjVoltage() —————> Returns voltage vector after baseline subtraction and inversion (in negative)
- vector<double>** GetTimeAdjVolt(**float** fraction) —————> Returns voltage vector after baseline subtraction, inversion if negative and time walk correction. Fixed time binning required for mean pulse shape. Time walk correction is performed by recalculation each voltage for the original time corresponding to the middle of the time bin by linear extrapolation of the two adjacent values.
- int** GetBasePos() —————> Returns the relative position of the waveform with respect to baseline, Not used

•Waveform Class

File: WaveForm.cxx

Main Functions

bool GetIsSignal()	double GetPedestalErr()
bool GetIsInWidow()	double GetCharge()
float GetTransimp()	double GetRiseTime(float bottom = -99, float top = -99)
float GetAmpGain()	double GetCFDTime(float fraction = -99)
int GetMaxIndx()	double GetTriggTime(float trigg = -99)
double GetMaxTime()	double GetdVdTMax()
double GetMax()	double GetdVdTCFD(float fraction = -99, int ndif = 0)
double GetMaxErr()	double GetCFDToT(float fraction = -99)
int GetMinIndx()	double GetTriggToT(float trigg = -99)
double GetMinTime()	double GetFrequency(int start = -I, int stop = -I)
double GetMin()	double GetJitterNdVdT()
int GetStrIndx()	double GetJitterRiseSNR()
int GetEndIndx()	double GetSignalFFT()
double GetNoise()	double GetNoiseFFT()
double GetNoiseErr()	bool dump()
double GetPedestal()	

•DUT Channel Class

File: DUTChannell.h

- Groups individual waveforms to a single object for more complicated calculations
- Inherits the `m_instrument`, and other global parameters
- Main Global Class variables:

✓ <code>m_board</code>	→ <code>enumerator</code>	<div>0 : SingleCh 1 : FourCh 2 : IN2P3 3 : CIVIDEC 4 : KU</div>
✓ <code>m_channelID</code>	→ <code>int</code> , Channel ID number	
✓ <code>m_channelName</code>	→ <code>TString</code> , Channel name (DUT name)	
✓ <code>m_ChRate</code>	→ <code>Long64_t</code> , Sampling rate	
✓ <code>m_ChTransimp</code>	→ <code>float</code> , Trans-impedance value of 1 st stage amp.	
✓ <code>m_ChAmpgain</code>	→ <code>int</code> , Second stage amp. Gain	
✓ <code>m_ChFraction</code>	→ <code>float</code> , CFD fraction for this channel	
✓ <code>m_ChTrigg</code>	→ <code>double</code> , trigger threshold for this channel (mV)	
✓ <code>m_ChNoPulses</code>	→ <code>int</code> , Number of added waveforms (events)	
<hr/>		
✓ <code>m_MeanChPulse</code>	→ <code>vector<pair<double, double>></code> , voltage vector of generated pulse template with uncertainties	

AqBoard TypDef

All variables are coupled with the standard `Get()` and `Set()` functions

•DUT Channel Class

File: DUTChannel.cxx

Class logic

Step 1, initialize: All class variables and pairs are initialized to -99, all vectors cleared at the beginning of the analysis.

Step 2, Append: Each new waveform is added to the channel. If waveform calculations have failed at some point, only successful results are added, rest filled with default values.

Step 3, Update: Channel Properties are updated and recalculated. Step to be performed at the end of the analysis.

Step 4, Print/Dump: Print all calculated info (if `m_verbose`)

Two methods of channel properties calculations

Fit method

- ✓ Add each property of each waveform to a histogram
- ✓ Generate a value for each property by fitting the histogram with an appropriate distribution

Template method

- ✓ Create waveform template from all added waveforms
- ✓ Pass the template waveform through the WaveForm class and recalculate all properties

•DUT Channel Class

File: DUTChannel.cxx

Fit Method

- For each added pulse add the computed values of properties given from the WaveForm Calss to a THI
- At the moment of **Update()**, apply the corresponding fit to the THI to extract the expected value of the property
- Fit type deepens on the property
- In case of fit failure, revert consecutively:

Gauss X Landau → **Gauss** → **Simple Mean / Standard deviation**

Quantity	Applied Fit type	
Min, Max voltage :	Gauss x Landau fit	
Start, stop, min , max indices :	Gaussian fit	
Noise / pedestal :	Gaussian fit Gaussian x Linear fit	} Dynamically defined Based on pedestal
Min, Max, Rise, Trigger time :	Gaussian fit	
Charge, dV/dT, Jitter, FFT :	Gauss x Landau fit	

•DUT Channel Class

File: DUTChannel.cxx

Template Method

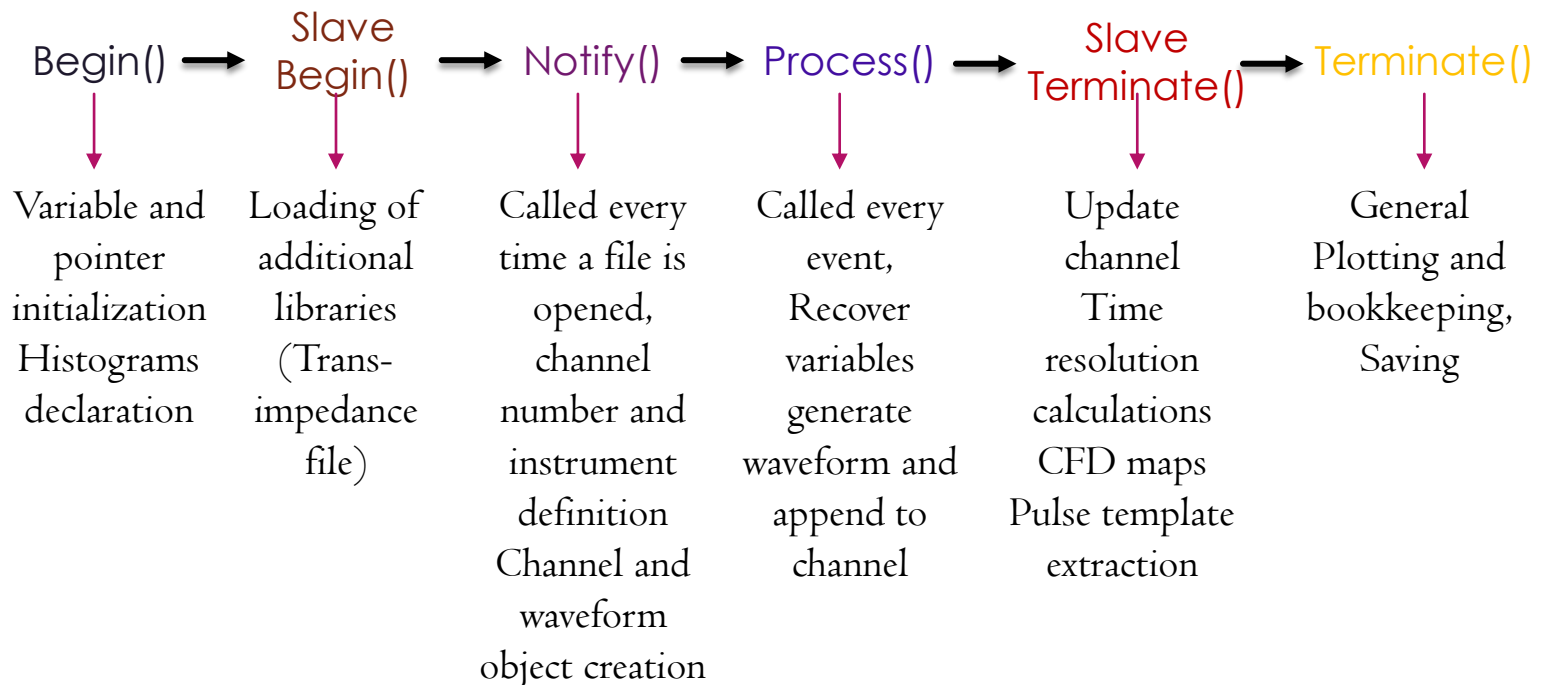
- For each added pulse recover the **time walk corrected** voltage values and add them to a TH2
 - For each bin apply a Gauss X Landau distribution and generate the MPV and uncertainty.
 - Create a pulse template from all the MPV values and feed it through the WaveForm Class
 - Recover all quantities computed on the template
-
- Both methods are performed!
 - Jitter computed through dV/dT and SNR in both approaches
 - SNR and Jitter statistical uncertainties are computed
 - Any differences between results can be considered systematic uncertainties

•LGADRun Class

File: HGTDRun.cxx

Main analysis method

- Organized in a TSelector Mode
- Main functions and execution order:

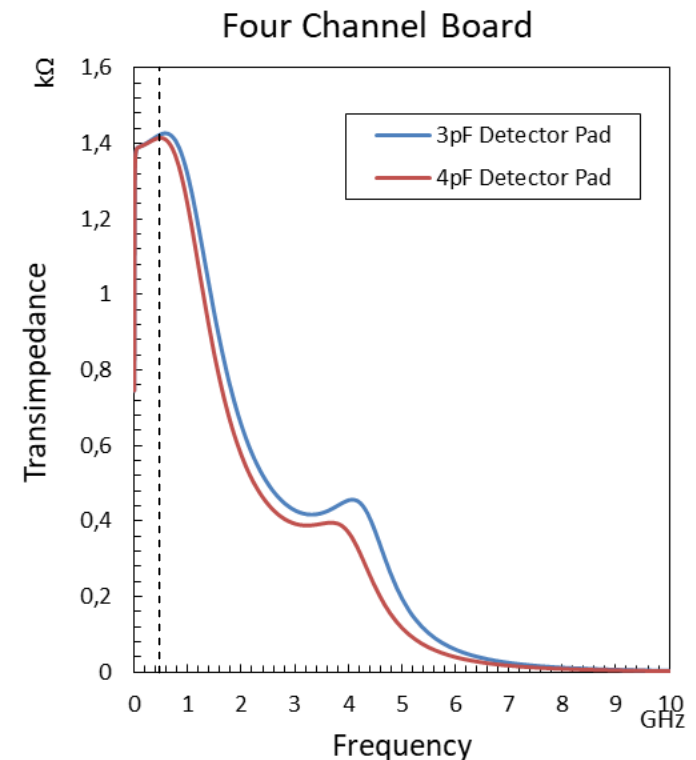
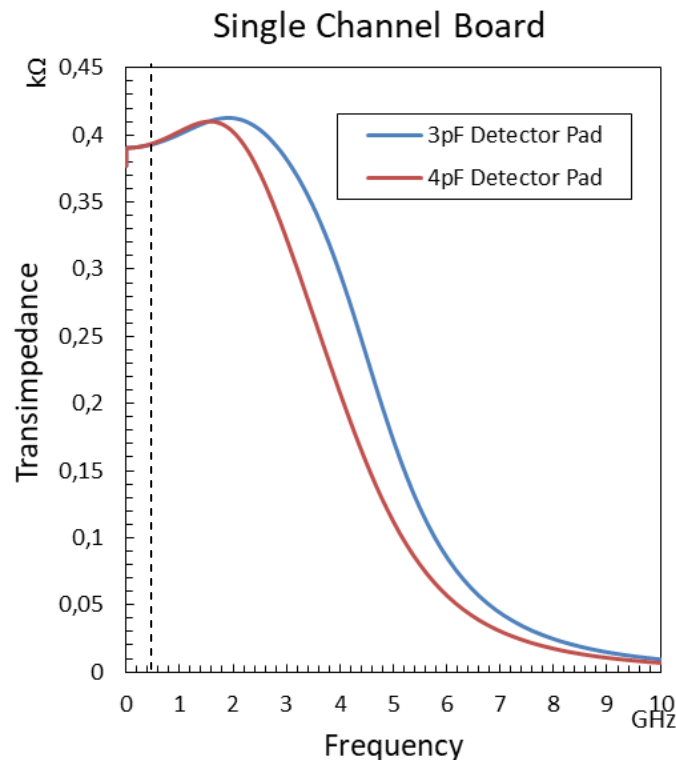


• Additional libraries

File: freq_trans.root

Trans - impedance simulation

- Points every 23.3 kHz in the 1 MHz – 10 GHz range
- 400 estimated values per simulation
- 2 detector pad capacitances, 3 and 4 pF



•Example

File: Run.C

Steering macro – Running example

```
1 {
2   gInterpreter->Reset(); // Reset Root
3
4   gInterpreter->AddIncludePath("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/");
5
6   gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Root/HGTDUtils.cxx+");
7   gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Root/HGTDUtils.cxx+");
8   gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Root/DataConverters.cxx+");
9   gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Root/HGTDSEl.cxx+");
10  gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Root/WaveForm.cxx+");
11  gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Root/DUTChannel.cxx+");
12  gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Root/HGTDRun.cxx+");
13
14  gROOT->ProcessLine("HGTDUtils* Utils = new HGTDUtils()");
15  gROOT->ProcessLine("Utils->SetVerbose(false)");
16
17  // Testbeam example
18  gROOT->ProcessLine("Utils->SetInstrument(2)");
19  gROOT->ProcessLine("Utils->SetInDataNames(\"/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Sample Datasets/Testbeam\", \"data_1536561052\")");
20  gROOT->ProcessLine("Utils->SetOutDataNames(\"/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Sample Datasets\")");
21  gROOT->ProcessLine("Utils->SetStartStopEvt(0, 0)");
22
23  // Lab example
24  gROOT->ProcessLine("Utils->SetInstrument(1)");
25  gROOT->ProcessLine("Utils->SetInDataNames(\"/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Sample Datasets/LecroyTXT\")");
26  gROOT->ProcessLine("Utils->SetOutDataNames(\"/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Sample Datasets\")");
27  gROOT->ProcessLine("Utils->SetStartStopEvt(2000, 0)");
28
29  // Sampic Example
30  // Start stop event not supported (no effect)
31  gROOT->ProcessLine("Utils->SetInstrument(0)");
32  gROOT->ProcessLine("Utils->SetInDataNames(\"/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Sample Datasets/Run_SAMPIC_Vagelis3_Data_2_26_2018_Ascii\")");
33  gROOT->ProcessLine("Utils->SetOutDataNames(\"/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Sample Datasets\")");
34
35  //Utils->ConvertData();
36
37  gROOT->ProcessLine("HGTDRun* Run1 = new HGTDRun(Utils)");
38
39  // Create the analysis chain
40  gROOT->ProcessLine("TChain *chain = new TChain(\"wfm\", \"\")");
41  gROOT->ProcessLine("chain->SetCacheSize(500 * 1024 * 1024)");
42  gROOT->ProcessLine("gEnv->SetValue(\"TFile.AsyncPrefetching\", 1)");
43
44  // Add data files to the chain
45  gROOT->ProcessLine("chain->Add(\"/afs/cern.ch/user/e/egkougo/private/HGTDUtils/Sample Datasets/data_1536561052.root\")");
46  gROOT->ProcessLine("chain->Process(Run1, \"\", 1)");
47 }
```

Include

Compile the
framework
with ACLiC

Create the object

Convert testbeam
data

Convert Lab
data

Convert
Sampic data

Create the Run

Create the chain

Run

•Deployment

5300 lines of code !!!

Main analysis method

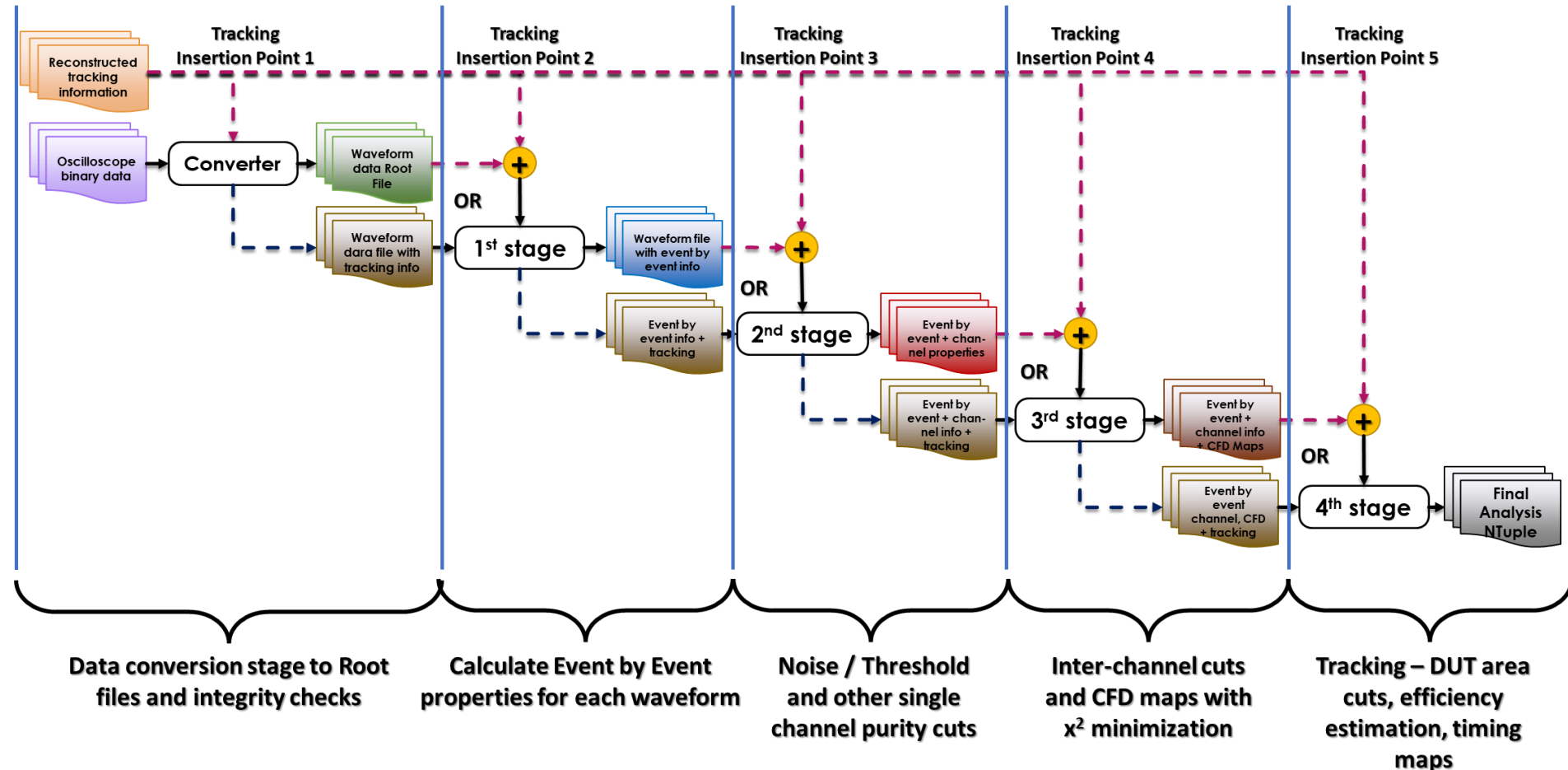
- Data conversion tested and verified in Testbeam, LeCroy Oscilloscope, Sampic, DRS4
- Compatible with ROOT 5/ ROOT 6 under investigation (FFTW libraries issues)
- Internal git (to IFAE + select beta testers), public once ROOT 6 issue resolved
- Lab results produced, test bema validation and pulse shape generation

Possible improvements and next steps

- Web-type interface for testbeam logbook connection
- Clear instruction Twiki
- Integration of second stage amplifier selection and option
- Board library with second stage trans-impedance and gain

•WfmTools

Analysis Roadmap



- Supported Tracking packages: Cory, EUTelescope, Bonn BTA
- 5 possible insertion points for tracking information merging

•File level table

Main analysis method

Input data file classification with a resume functionality from every analysis level to any higher level:

- **Level 0** → Converter oscilloscope data file to root only
 - **Level 1** → Event by event properties
 - **Level 2** → Event by event properties + main channel Fits
 - **Level 3** → Event by event analysis, channel properties fits, time resolution
CFD maps, channel to channel cuts and chi2 minimization for
working point
 - **Level 4** → Tracking level analysis with exclusion areas
-

Data Converter Support Table				
Instrument	Tracking	Multiple Runs	Multiple Units	Start Event ≠ 0
Sampic	YES	YES	NO	YES
LabTXT	NO	NO	NO	YES
TektronixTXT	YES	YES	NO	NO
LeCroy Binary	YES	YES	YES	NO
Agilent/Tektronix Binary	YES	YES	YES	NO

•Analysis Setup

Input Filenames

```
void LGADBase::SetInDataNames(TString DataDir, TString DataName, TString ext)
```



Directory



Filename



Extension

➤ **Initialization :** All three initialized to ""

➤ Case Diagram

Directory has full path with filename and extension. All other arguments ignored; file specified in the directory is analyzed

Directory has no extension. Considered as directory containing analysis files

Filename Argument Specified

Filename Argument Empty

Filename contains extension. Combined with directory it consists analyzed file. Subsequent arguments ignored.

Filename contains no extension.

Extension provided. All files in Directory with Extension added for analysis

Extension not provided. All files in Directory added for analysis

Extension argument provided. All files starting with filename followed by any variation of extension added for analysis

No extension specified. All files in Directory who's name starts with Filename are added for analysis