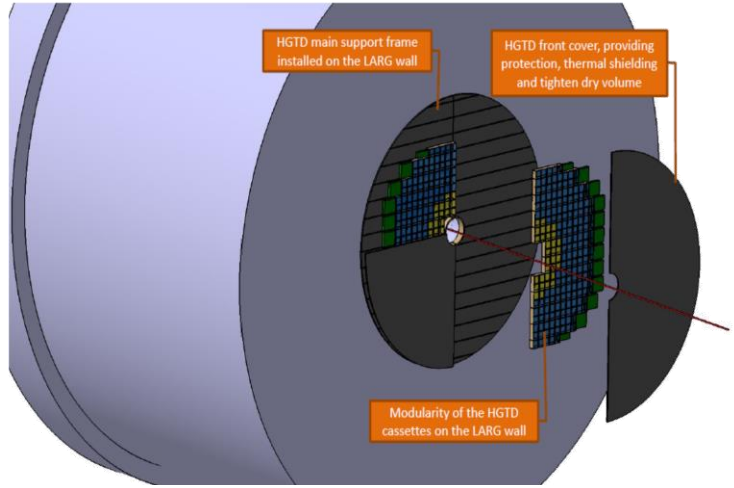


Timing analysis framework

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:

- HGTDUtils
  - cmt → RootCore implementation and version control
  - HGTDUtils → Package headers
  - macros → Steering macro examples (local and grid)
  - Root → Classes implementation
  - share → External libraries

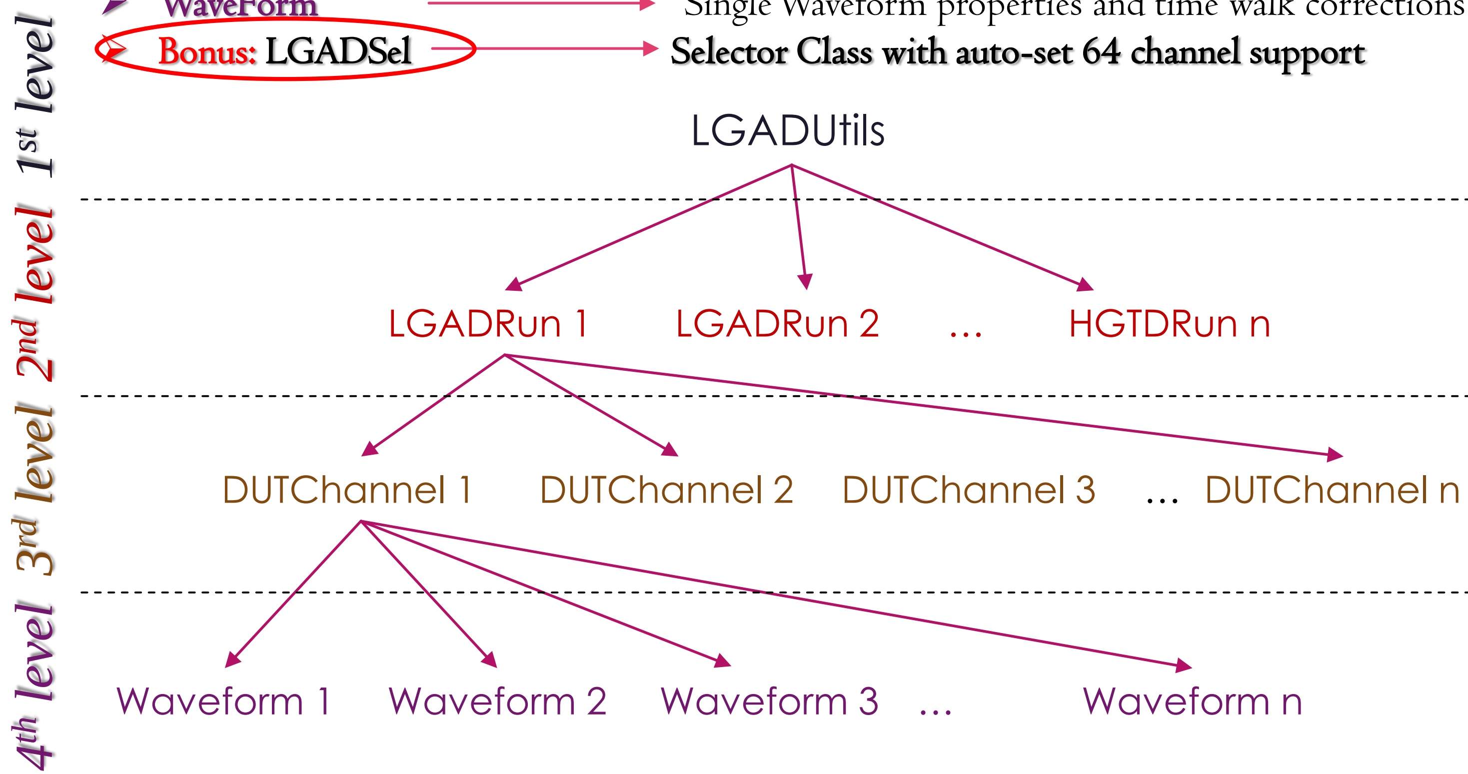
- Following standard ATLAS analysis package organization
- Running as interpreted (CINT), compiled (CMake) or RootCore version
- Tested in multiplatform environment (Windows, Linux), appropriate preprocessor instructions incorporated for compatibility
- Only requirement ROOT (with **FFTW** and **RooFit**), 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?

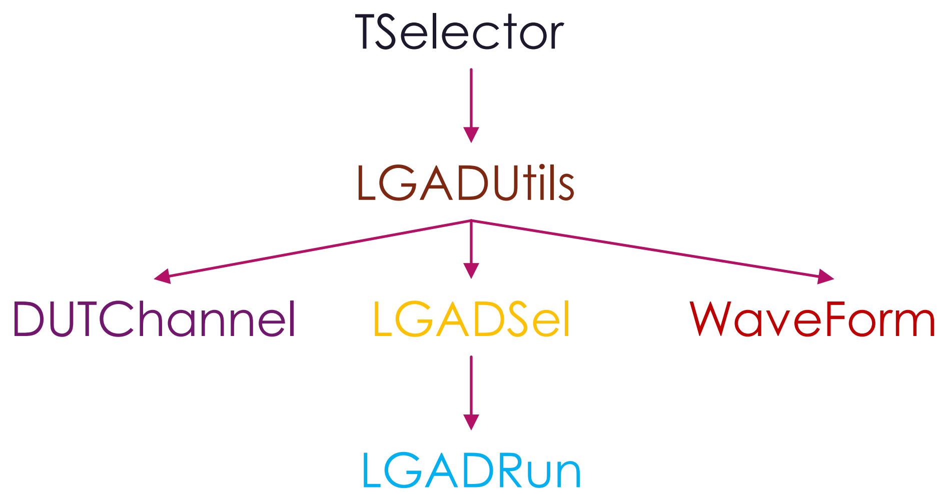
- Four main classes with dedicated header and implementation files, one wrapper class handling user interaction
  - LGADBase → Wrapper to handle user I/O and pass arguments
  - LGADUtils → Basic framework function and infastructure
  - LGADRun → Timing resolution, CFD maps, multi DUT operations
  - LGADChannel → 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**



•Timing resolution Framework

C++ 11

Inheritance



- Inheritance from a TSelector allows for seamless analysis with interal 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

File: HGTDUtils.h

- Public Variables

Global Steering variables

✓	m_instrument	enumerator (AqIsntrument TypDef): 0 : Sampic (ASCI); 1 :LeCroy Scope (ASCI); 2 : Agilent Scope (Binary); 3 : Tektronix Scope (ASCI); 4 : LeCroy Scope (Binary)
✓	m_verbose	bool, Activate debug messages
✓	m_ofname	TString, Output root file name
✓	m_TransFile	TString, Transimpedance simulation library
✓	m_ofile	TFile*, output root file universal pointer
✓	m_tree	TTree*, analysis tree universal pointer
✓	m_event	unsigned int, total number of events
✓	m_nchan	unsigned int, number of active channels
✓	m_TrackComb	bool, enables or disables combination with tracking

Propagated to analysis

✓	m_srate	vector<Long_t64>, Sampling rate per channel
✓	m_npoints	vector<unsigned int>, no. of points per channel
✓	m_scale	vector<double>, Voltage scale per channel
✓	m_physt	vector<double>, Physical cell time → Sampic
✓	m_ordrt	vector<double>, Ordered cell time → Sampic
✓	m_triggTime	vector<double>, Trigger time → Agilent Scope, Tektronix Scope

•HGTDUtils Class

File: HGTDUtils.cxx

Public functions

➢ Constructor and Destructor

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

virtual ~LGADUtils()

➢ 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) → Retunes sampling rate for specified channel

unsigned int GetNPoints(**unsigned int** ch) → Retunes 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, **const string** fill, **const string** whitespace) → Replaces or removes characters in a string with the specified sequence in "fill". Default is empty.

string trim(**const string** str, **const string** whitespace) → Removes characters or spaces from the beginning and end of the string

unsigned int CountFiles(**const char*** dir, **const char*** ext) → Returns number of files of extension "ext" in folder "dir". POSIX dependent.

int DirExists(**const char*** path) → Checks if folder exists. Returns 1 if it does, 0 if not.

Supported types: int, bool, double, float

template Derivate(**template** *T, **int** start = 1) → Analytically computes point derivative of vector w starting at index start.

double Mean(**template** *T, **int** start = -1, **int** stop = -1) → Compute standard mean value of vector W from element start to stop

double Stdev(**template** *T, **int** start = -1, **int** stop = -1) → Compute standard deviation of vector w form element start to stop

•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 Ntpule with respect to instrument and channels

•LGADUtils Class – Data Converters

File : DataConverters.cxx

Data Converters

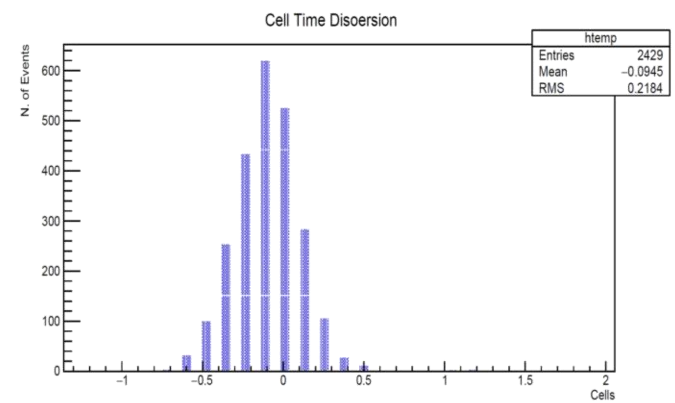
➢ Dedicated Functions

bool ConvertData() → Starts the data conversion with the parameters set. m_instrument and input/output dirs. Must be already defined

void SetStartStopEvnt(**unsigned int** Evntt1, **unsigned int** Evnt2) → Set the event to start the conversion from and where to stop. If not called, it goes thought 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 peeked at 0*
- *Bin size is equal to sampling rate*



Correction

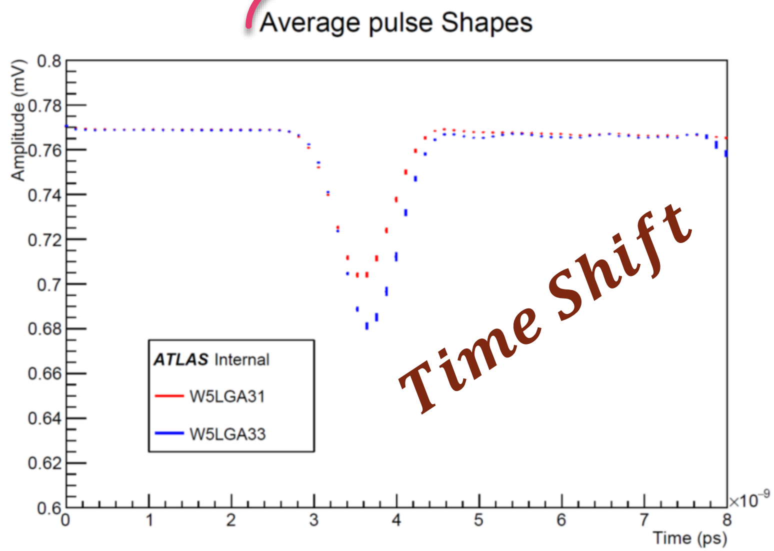
Time for each channel needs to be recalibrated with respect to a channel of reference

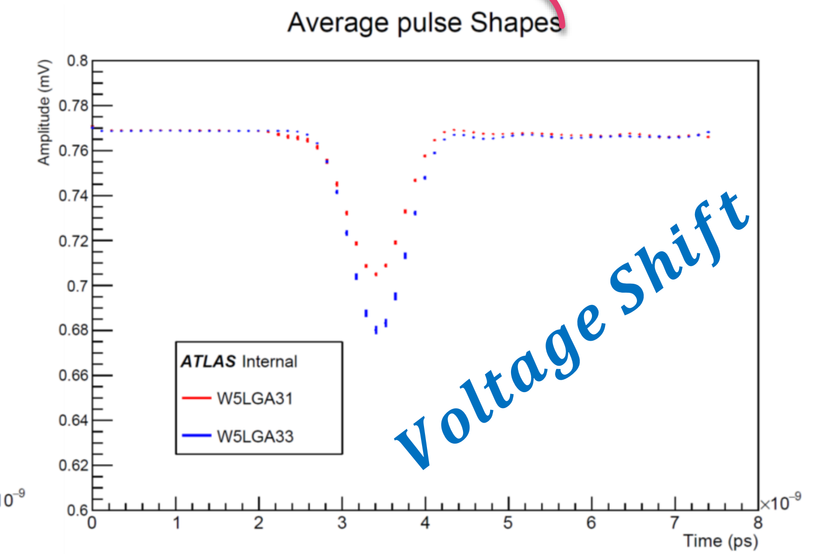
•LGADUtils Class – Data Converters

Time Alignment

File : DataConverters.cxx

Tow possible correction methods





- 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

•LGADUtils Class - HGTDFits

File: HGTDFits.cxx

RooFit Implementation

```
int GaussFit(vector<double> *w,
             pair<double, double> &gmean,
             pair<double, double> &gsigma,
             pair<int, int> points)
```
Performs Gaussian fit on vector w, returns mean and sigma with uncertainties. Points sets the start and stop index of fiting

```
int LinearFit(vector<double>* vec,
              pair<double, double> &slope,
              pair<double, double> &intersept)
```
Performs Linear fit on w, returns slope and intercept with errors. Fit limits adjusted automatically.

```
int GauLinear(std::vector<double>* vec,
              pair<double, double> &magMPV,
              pair<double, double> &magSigma)
```
Performs Gauss X Linear fit on w, returns mean and sigma with uncertainties.

```
int GauLandauFit(vector<double>* vec,
                 pair<double, double> &magMPV,
                 pair<double, double> &magSigma)
```
Performs Gauss X Landau fit on w, returns MPV and sigma with uncertainties.

```
float LinearInter(float x1, float y1,
                  float x2, float y2, float y3)
```
Tow point linear interpolation on y3, between (x1, y1) and (x2 y2).

```
double FFT(vector<double> *w, int snrate,
           int start, int stop)
```
Fast Furrier transform on w, from point start to stop. It returns the first order frequency.

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 nTpule branches
- Shell class for the analysis, written in the HGTDRun class

•LGADSelector Class

Ntuple Variables

File: HGTDSel.h

General

✓ EvnNo	unsigned int, event number
✓ w0I - wXX	double, channel amplitude in millivolts
✓ t0I – tXX	double, channel time in picoseconds

Agilent, LeCroy, Tektronix

✓ vScale0I – vScaleXX	float, voltage sale per channel
✓ nPoints0I – nPointsXX	unsigned int, number of points per channel
✓ SnRate0I – SnRateXX	Long_64t, Sampling rate per channel

Agilent

✓ triggTimeXX – triggTimeXX	double, trigger time per channel

Lecroy, Tektronix

✓ Triggtime	double, global trigger time

Sampic

✓ phystXX - phystXX	double, time of the physical cell of the chip
✓ ordrtXX - ordrtXX	double, time with respect of the general clock
✓ nPoints	double, global number of points per channel
✓ SnRate	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 typdef
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

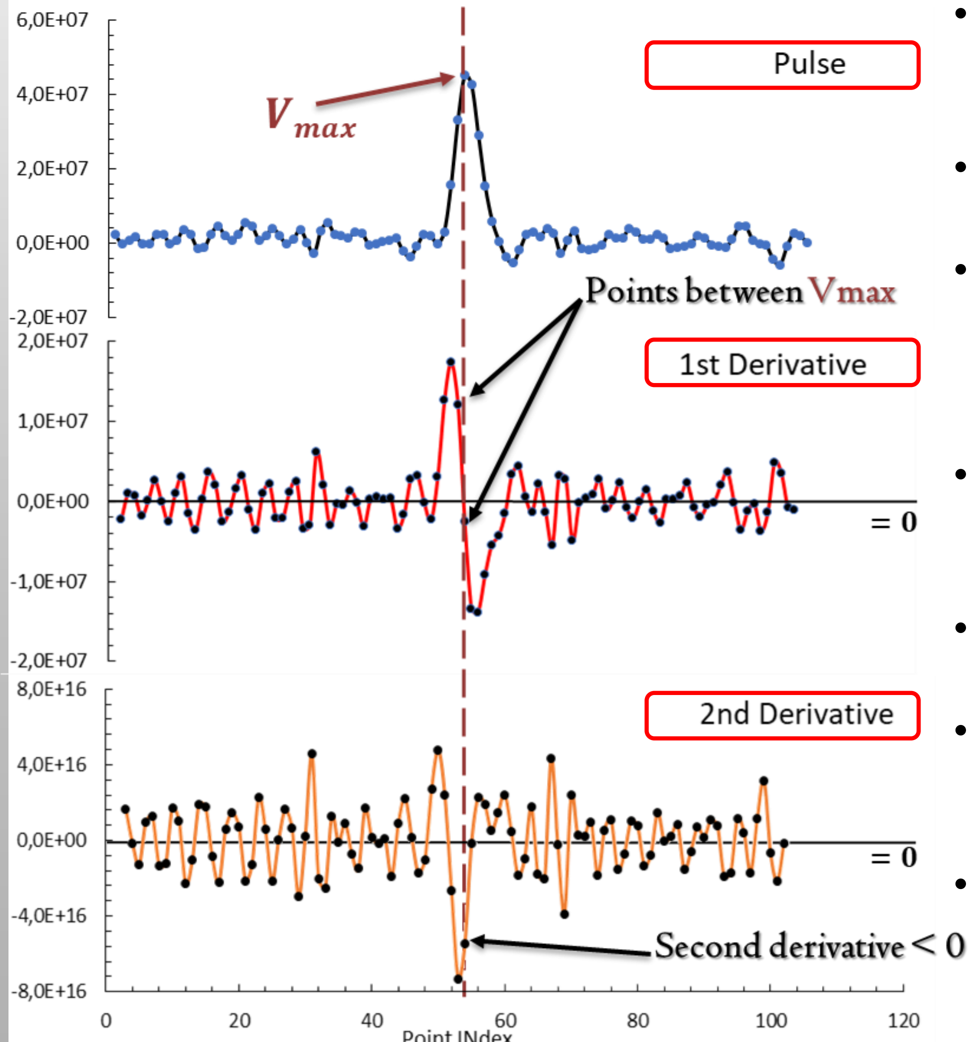
File: WaveForm.cxx

Polarity and Peak detection algorithm

Caution:

No smoothing applied!!

Tow point linear interpolation for precision and to satisfy continuity requirement of local maxima mimima 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 $SD_{maxima} < 1.05 * SD_{minima}$
  $D_m(max) > D_m(min)$

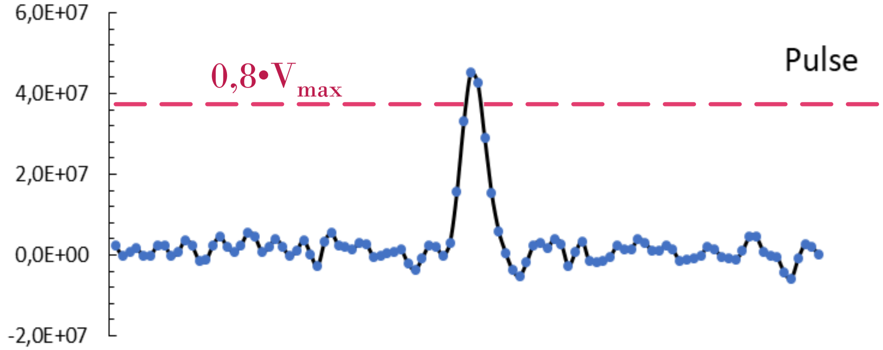
Positive polarity

•Waveform Class

File: WaveForm.cxx

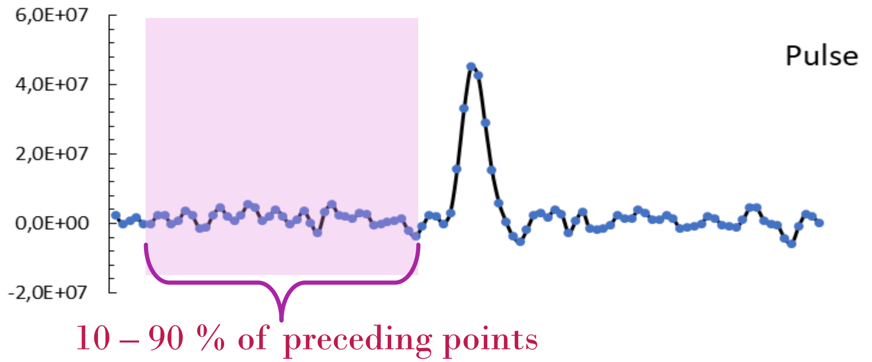
Noise

Is Signal?



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

Noise Points



- Noise calculation include only points before pulse
- Points considered:

  N_{stsrt}*10% to N_{stsrt}*90%
- If pulse not within window perform same operation after N_{end}

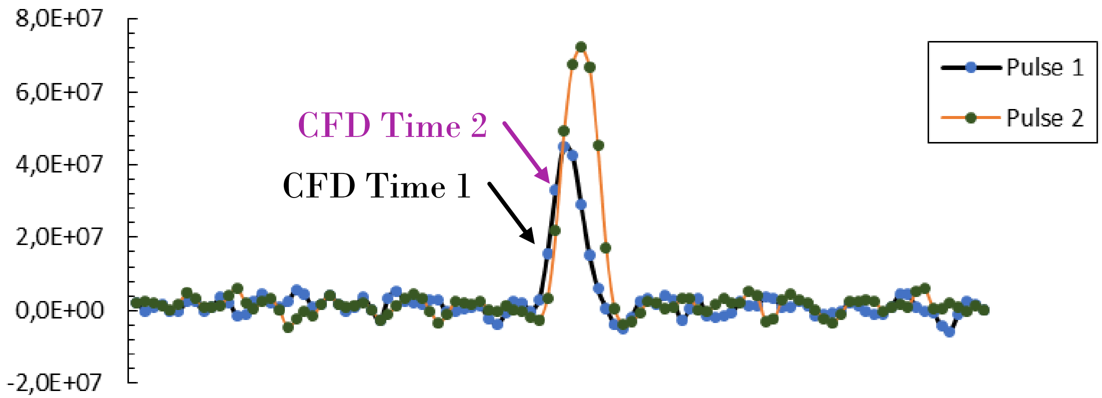
Noise estimation

- 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 $t_0 = t_{CFD}$
  - Readjust time of each point as $t' = t - t_{CFD}$

Simples, 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 od
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()
bool GetIsInWidow()
float GetTransimp()
float GetAmpGain()
int GetMaxIndx()
double GetMaxTime()
double GetMax()
double GetMaxErr()
int GetMinIndx()
double GetMinTime()
double GetMin()
int GetStrIndx()
int GetEndIndx()
double GetNoise()
double GetNoiseErr()
double GetPedestal()
```

```
double GetPedestalErr()
double GetCharge()
double GetRiseTime(float bottom = -99, float top = -99)
double GetCFDTime(float fraction = -99)
double GetTriggTime(float trigg = -99)
double GetdVdTMax()
double GetdVdTCFD(float fraction = -99, int ndif = 0)
double GetCFDToT(float fraction = -99)
double GetTriggToT(float trigg = -99)
double GetFrequency(int start = -1, int stop = -1)
double GetJitterNdVdT()
double GetJitterRiseSNR()
double GetSignalFFT()
double GetNoiseFFT()
bool dump()
```

•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:

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

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

•DUT Channel Class

File: DUTChannell.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 thought the WaveForm class and recalculate all properties

•DUT Channel Class

File: DUTChannell.cxx

Fit Method

- For each added pulse add the computed values of properties given form the WaveForm Calss to a TH1
- At the moment of Update(), apply the corresponding fit to the TH1 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
Min, Max, Rise, Trigger time :	Gaussian fit
Charge, dV/dT, Jitter, FFT :	Gauss x Landau fit

Dynamically defined Based on pedestal

•DUT Channel Class

File: DUTChannell.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 trough the WaveForm Class
- Recover all quantities computed on the template

- Both methods are performed!
- Jitter computed though dV/dT and SNR in both approached
- 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:

Begin() → Slave Begin() → Notify() → Process() → Slave Terminate() → Terminate()

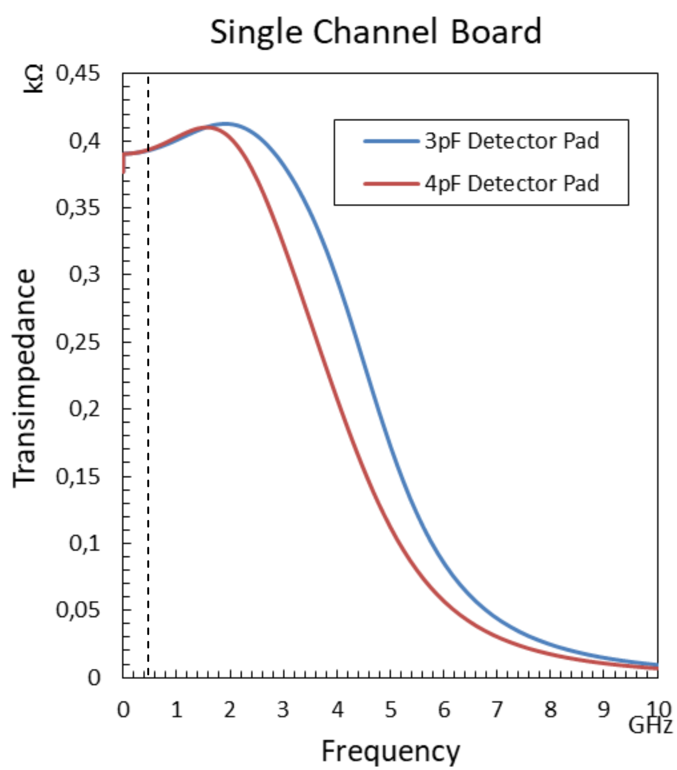
Begin()	Slave Begin()	Notify()	Process()	Slave Terminate()	Terminate()
Variable and pointer initialization Histograms declaration	Loading of additional libraries (Trans-impedance file)	Called every time a file is opened, channel number and instrument definition Channel and waveform object creation	Called every event, Recover variables generate waveform and append to channel	Update channel Time resolution calculations CFD maps Pulse template extraction	General Plotting and bookkeeping, Saving

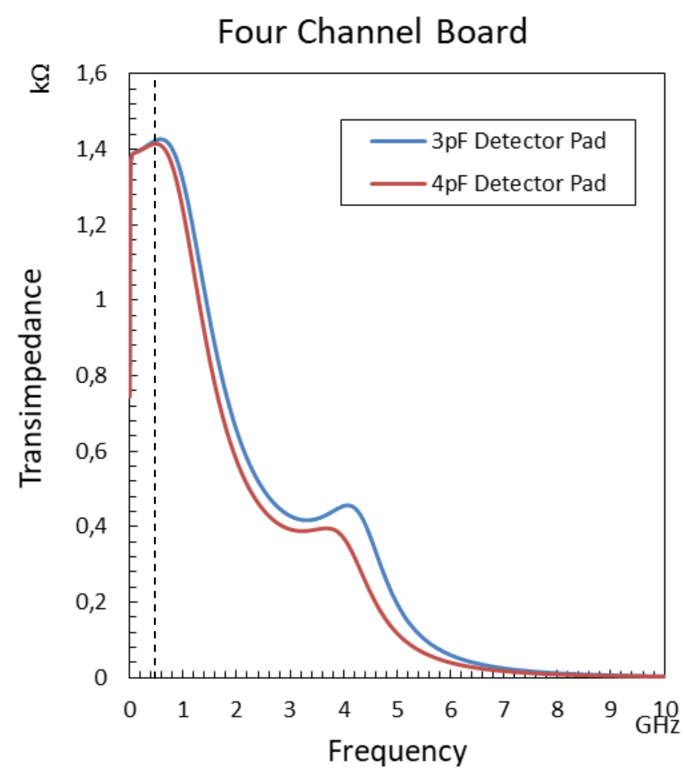
•Additional libraries

Trans - impedance simulation

File: freq_trans.root

- 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

```
{
    gInterpreter->Reset(); // Reset Root

    gInterpreter->AddIncludePath("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/");

    gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Root/HGTDUtils.cxx+");
    gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Root/HGTDFits.cxx++");
    gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Root/DataConverters.cxx+");
    gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Root/HGTDSel.cxx+");
    gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Root/WaveForm.cxx++");
    gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Root/DUTChannel.cxx+");
    gInterpreter->LoadMacro("/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Root/HGTDRun.cxx+");

    gROOT->ProcessLine("HGTDUtils* Utils = new HGTDUtils()");
    gROOT->ProcessLine("Utils->SetVerbose(false)");

    // Testbeam example
    gROOT->ProcessLine("Utils->SetInstrument(2)");
    gROOT->ProcessLine("Utils->SetInDataNames(\"/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Sample Datasets/Testbeam\", \"data_1536561052\")");
    gROOT->ProcessLine("Utils->SetOutDataNames(\"/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Sample Datasets\")");
    gROOT->ProcessLine("Utils->SetStartStopEvnt(0, 0)");

    // Lab example
    //gROOT->ProcessLine("Utils->SetInstrument(1)");
    //gROOT->ProcessLine("Utils->SetInDataNames(\"/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Sample Datasets/LecroyTXT\")");
    //gROOT->ProcessLine("Utils->SetOutDataNames(\"/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Sample Datasets\")");
    //gROOT->ProcessLine("Utils->SetStartStopEvnt(2000, 0)");

    // Sampic Example
    // Start stop event not supported (no effect)
    //gROOT->ProcessLine("Utils->SetInstrument(0)");
    //gROOT->ProcessLine("Utils->SetInDataNames(\"/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Sample Datasets/Run_SAMPIC_Vagelis3_Data_2_26_2018_Ascii\")");
    //gROOT->ProcessLine("Utils->SetOutDataNames(\"/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Sample Datasets\")");

    //Utils->ConvertData();

    gROOT->ProcessLine("HGTDRun* Run1 = new HGTDRun(Utils)");

        // Create the analysis chain
    gROOT->ProcessLine("TChain *chain = new TChain(\"wfm\", \"\")");
    gROOT->ProcessLine("chain->SetCacheSize(500 * 1024 * 1024)");
    gROOT->ProcessLine("gEnv->SetValue(\"TFile.AsyncPrefetching\", 1)");

    // Add data files to the chain
    gROOT->ProcessLine("chain->Add(\"/afs/cern.ch/user/e/egkougko/private/HGTDUtils/Sample Datasets/data_1536561052.root\")");
    gROOT->ProcessLine("chain->Process(Run1,\"\",1)");
}
```

Include

Order matters!!

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