JOINT INSTITUTE
FOR NUCLEAR RESEARCH

JINR Summer Student Programme 2016

Development of the data preprocessing complex for the ToF-400 system in the BM@N experiment

JINR Summer Student Programme project report

Author:
Kurganov Alexander
Russia, Moscow state university
Faculty of physics

Supervisors:
Vadim Babkin
Mikhail Rumyantsev
Vyatcheslav Golovatyuk

Participation period:
10 July - 15 August

2016

Contents

1 Abstract **2**

2 Intro **4**

2.1 ToF-400 system brief description 4

2.2 Front-end and readout electronics 5

2.3 Aim of the work 6

3 Realization and methods **6**

3.1 BmnRoot framework and class structure 6

3.2 Mapping 8

3.3 INL Correction 11

3.4 Complete first conversion stage algorithm and determination of the signal amplitude 12

3.5 Complete second conversion stage algorithm 14

4 Conclusion **16**

5 References **17**

Appendices **18**

A Corrected mapping **18**

B Classes and structures description **20**

B.1 BmnTDCDigit 20

B.2 BmnTof1Digit 20

B.3 BmnTof1Raw2Digit 21

B.4 BmnTof1TDCParameters 21

B.5 BmnTof1Map2 21

B.6 myTof1Hit 23

B.7 TofHitConverter 23

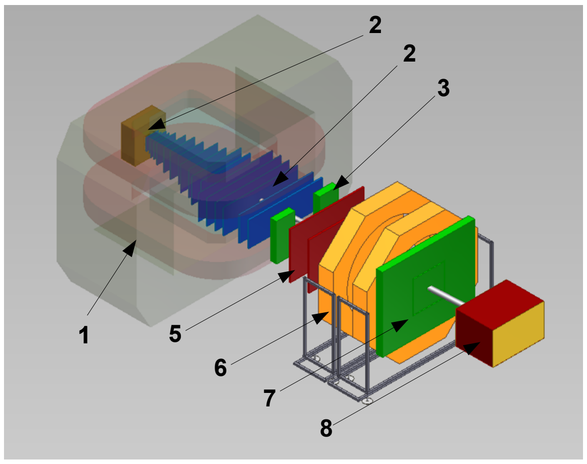
C Complete list of built debug histograms **25**

1 Abstract

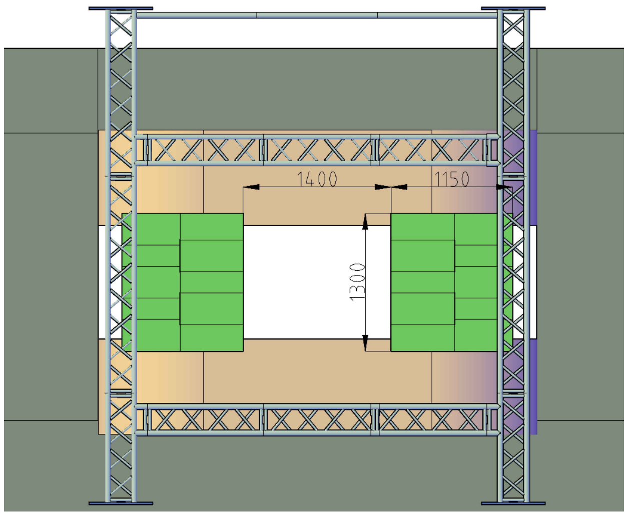
JINR Nuclotron will provide the heavy-ion beams with energies up to 6 A·GeV for isospin symmetric nuclei and 4.65 A·GeV for Au nuclei. In the central heavy-ion collisions at these energies, the nuclear densities of about four times nuclear matter density can be reached. These conditions are well suited to investigate the equation-of-state of dense nuclear matter, which plays a central role for the dynamics of core collapse supernovae and for the stability of neutron stars.

Also the heavy-ion collisions are a rich source of strange particles. The extension of the experimental program is related to the study of in-medium effects for vector mesons and strangeness decaying in hadronic modes.

For these purposes, it was proposed to install the BM@N experiment in the fixed-target hall of Nuclotron with the final goal to perform a research program focused on the production of strange matter in the heavy-ion collisions at beam energies between 2 and 6 A·GeV. [1][2]



(a) BM@N experiment schematic



(b) ToF-400 detector schematic from another view with basic dimensions

Figure 1: BM@N experiment and ToF-400.

The experiment combines high precision track measurements with time-of-flight information for particle identification and uses total energy measurements for the analysis of the collision centrality. The experiment setup consists of (see fig. 1a) several subsystems [2]:

- Analysing magnet **(1)**. The gap between the poles of the analyzing magnet is around 1 m. The magnetic field can be varied up to 1.2 T to get the optimal BM@N detector acceptance and momentum resolution for different processes and beam energies.

- Two coordinate planes of Gaseous Electron Multipliers (GEM) detectors **(3)** located in the magnetic field, straw **(5)** and drift chambers **(6)** located outside it are used to measure the charged track momentum and multiplicity.

- Two multi-gap resistive plated chambers time-of-flight detectors with a strip readout - TOF400 **(4)** and TOF700 **(7)**. The design of these detectors allows to discriminate between hadrons (π, K, p) as well as light nuclei with the momentum up to few GeV/c produced in multi-particle events.

- T0 detector **(2)**, which is planned to trigger central heavy ion collisions and provide a trigger (T0) signal for the TOF400 and TOF700 detectors.

- Zero degree calorimeter **(8)** (ZDC) designed for the analysis of the collision centrality by measuring the energy of forward going particles.

This work will be focused on the TOF400 detector (fig. 1a **(4)**, 1b) scientific data preprocessing complex development.

2 Intro

2.1 ToF-400 system brief description

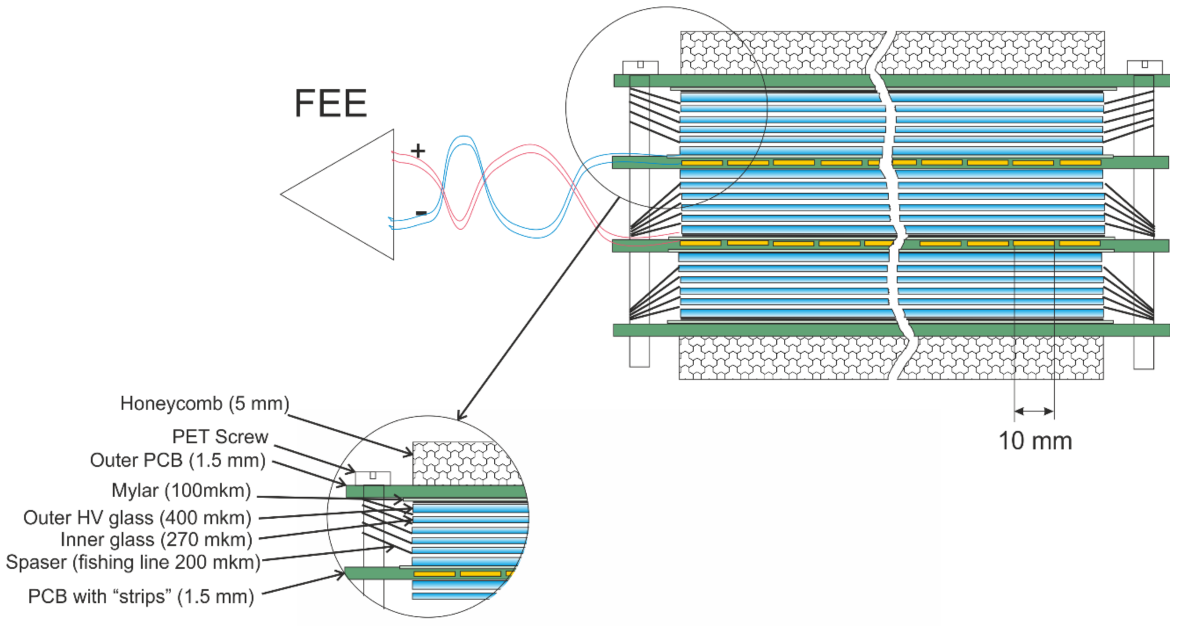


Figure 2: The ToF-400 plane schematic

The ToF-400 system is symmetrical relative to the beam and consists of two parts on each (left and right) side of the beam. Both of these parts consist of two gas boxes with 5 mRPC detector planes in each, which overlap a bit. The total number of planes in the completed ToF-400 system will be $2 \times 2 \times 5 = 20$. Each of the planes has 48 strips in it.[3]

A ToF-400 detector plane consists of three stacks with 5 gas gaps in each. Common float glass is used as resistive electrodes. The pickup electrodes look like strips and made on a printed circut board (PCB); differential analog signal from both sides of the strip is transferred to front-end electronics. Transferring signals from both sides of the strip enchanses the time resolution and provides a way to measure the coordinate along the strip by measuring the time difference between the signals from two sides of the same strip.

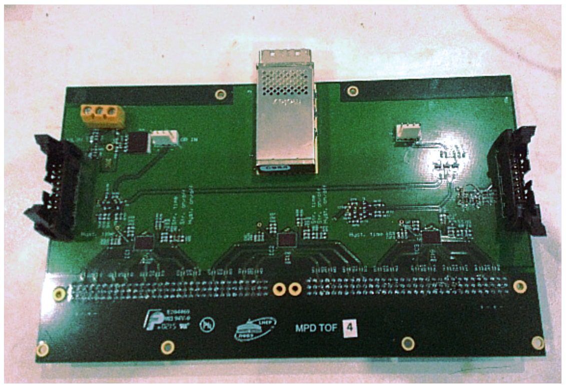
Currently in the ToF-400 three detector planes are installed in one gas box and experimental run data from them is used for debugging and testing purposes. Hereinafter these will be enumerated as planes 1, 2, 3, starting from the actual bottom in the experimental setup (so plane 2 is the center plane which overlaps a bit with the plane 1 and plane 3).The data preparation complex, which will be developed in this work, must be easily expandable and be ready to prepare raw data from all of the 20 future detector planes.

2.2 Front-end and readout electronics

A fast front-end preamplifier discriminator chip NINO[4], originally developed for ALICE experiment TOF system, is used in BM@N Time-of-Flight system. The NINO chip has 8 channels, each of which has an ultra-fast preamplifier with a peaking time less than 1 ns, discriminator with a minimum detection threshold of 10 fC and output stage which provides s pulse LVDS output signal. Width of the pulse is determined by the duration of the period of time, in which the signal is greater than a modifiable threshold, which is set once in a run. Using this time-over-threshold technology one can measure the signal amplitude.

The preamplifier PCB has three NINO chips on it; so, each preamplifier board has $3 \times 8 = 24$ channels. To measure the signal on both sides of each strip in the system, $(48/24) \times 2 = 4$ preamplifier boards are connected to each plane. On each preamplifier PCB two 3-row, 32-column connectors are placed, using which the preamplifier board connects to a special "transmitter" PCB on the gas box, one side of which is located inside the gas box (and has 3 connectors with 8 strips connected to each) and the other on the outside.

The signal from the strip (on the input of the preamplifier board) has a considerably long front period duration. The duration of period of time, in which the signal voltage rises from zero to the threshold voltage is dependent from the signal overall amplitude; due to this, the time resolution is spoiled. To overcome this effect, a time-amplitude correction must be applied to the experimental data. The LVDS signal is then transmitted to the TDC72VHL time-to-digital


(a) Preamplifier board

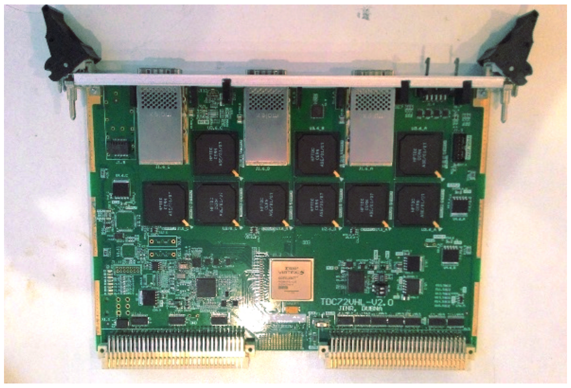

(b) TDC72VHL PCB

Figure 3: Front-end electronics and TDC PCBs

converters (TDCs), made in a VHL module form. Each TDC72VHL module has 9 HPTDC[5] 32-channels chips on the PCB. Each HPTDC chip has 1024 bins, and is operating in the 24ns mode ($24ns/1024 \approx 23.4ps$ per bin). To provide a way to work in this mode, the HPTDC chip's channels have to be combined by quads, so in total each HPTDC has only 8 “real” channels instead of 32. Unfortunately, operating in this mode leads to larger nonlinearity, so corrections using integral nonlinearity (INL) measurements should be made while preprocessing the experimental data. These INL characteristics vary not only from chip to chip, but also from channel to channel.

2.3 Aim of the work

The aim of this work is to develop a scientific data preprocessing complex, which will prepare raw TDC experimental data for any future analysis. This system should include:

- INL (integral nonlinearity) corrections;
- Conversion of TDC72VHL board serial number and the triggered channel index on it to strip number, it’s side and plane ID (mapping);
- Measuring the actual time between leading and trailing parts of the signal (signal amplitude), as the TDC72VHL boards output two distinct signals on the signal leading and trailing parts;
- Combining the measurements on both sides of the strip to determine the position along the strip and increase the overall time resolution;

3 Realization and methods

3.1 BmnRoot framework and class structure

The BM@N experiment uses a special software framework BmnRoot[6] to process and analyse the experimental data. This framework is build around the ROOT[7] environment and the FairSoft object-oriented framefork FairRoot[8]. Overall the BmnRoot is a powerfull tool for BM@N detector performance studies, development of algorithms for reconstruction and physics analysis of the data. It also provides an invterface for the PostgreSQL database, which will be used in this work to download the nessessary data. The BmnRoot framework is written in C++.

The Tof-400 data preprocessing complex was developed during this work inside the BmnRoot framework. To perform such an integration, a couple of C++ classes’ source codes in the BmnRoot were changed and a couple of classes were added.

To perform a decoding process from raw binary experimental data to a ROOT TTree file, the class **BmnRawDataDecoder** is included in the framework. It performs the decoding process in two stages:

1. Decoding the raw binary data and putting it in a couple of different structures, which then are inserted inside a ROOT TTree structure and saved. This process only saves the ADC and TDC data in a bit more convenient format than binary and does not perform any changes to the data.

2. Conversion of the raw TDC and ADC measurements to the detector data with applied INL corrections and mapping, saving it to a different TTree file.

The TDC information after the first stage is stored in a **BmnTDCDigit** class. The **BmnTDCDigit** class stores only one TDC digit (TDC measurement) and has the following fields inside:

- Crate serial and slot, in which the current VHL module is inserted;
- VHL module type ($0x12$ for the TDC72VHL TDC);
- HPTDC chip's id;
- HPTDC chip channel index;
- Type of the TDC digit: leading or trailing;
- Time (value) of the digit in TDC bins.

During the second conversion the **BmnRawDataDecoder** class uses the **BmnTof1Raw2Digit::FillEvent** method to convert an array of **BmnTDCDigit** structures into Tof-400 data. The Tof-400 data after the second stage is stored in a **BmnTof1Digit** structure. The **BmnTof1Digit** structure stores the plane id, strip index, side of the strip, leading time of the signal and its width.

Only the second stage of the conversion process concerns this work. As said before, the **BmnTof1Raw2Digit** class is used to convert an array of **BmnTDCDigit** structures in the event into an array of **BmnTof1Digit** structures. During this work, the **BmnTof1Raw2Digit** class was completely remade to perform the convertion process properly and apply the INL corrections and mapping during it. This classes' description can be found in the appendix B.

After these two stages, an additional conversion process was added along with a couple of classes. During this stage an array of **BmnTof1Digit** structures is converted in an array of **myTof1Hit** structures. This conversion stage is needed to “connect” two **BmnTof1Digit** instances on different sides of one strip into one **myTofHit** structure instance. The **myTof1Hit** class stores the signal amplitude, average leading time, position along the strip in time equivalent (difference between the leading time of two TDC digits), and the plane and strip IDs. The **myTof1Hit** class was added to the BmnRoot framework during this work.

To perform the last described conversion, a new class **TofHitConverter** was created. Its description also can be found in the appendix B.

After the last conversion, some data analysis should be performed to check if every step is done properly and to measure the time-amplitude dependence. A couple of debugging histograms were also built inside the **TofHitConverter** class right during the conversion process. These histograms will be described later.

In conclusion, the data preprocessing complex itself consists of two main classes **BmnTof1Raw2Digit** and **TofHitConverter** each representing a distinct conversion stage. The first stage's input data is an array of **BmnTDCDigit** structures in the event, the output data is an array of **BmnTof1Digit** instances. The second stage's input data is the first stage output, and the output data of the second stage is an array of **myTof1Hit** structures. During the first stage, INL corrections, mapping appliance and signal width measurements are performed. During the second stage the data from two different sides of the same strip is combined.

In further subsections the full conversion process stages will be described in detail.

3.2 Mapping

Mapping in the developed program is applied during the first stage of full conversion process, applied in the **BmnTof1Raw2Digit** class method and consists in fact of three conversion processes:

1. Index of the HPTDC chip in it and triggered channel of the HPTDC to TDC global (across all of the HPTDC chips on it) channel (hereinafter TDC channel);

2. Crate ID (the sync module ID in it) and insertion slot of the TDC in the crate to TDC serial number;

3. TDC serial number and the TDC channel to plane ID, strip index and its side.

The first conversion is quite straightforward and is performed using a simple equation

$$TDC\ Channel = (HPTDC\ index) \cdot 8 + (HPTDC\ channel) \tag{1}$$

Second and third conversions are made using special tables, which are downloaded from a PostgreSQL database and then stored in RAM using std::map key-value containers. In second conversion, std::pair of crate ID and slot number is used as a searching key and the TDC serial is used as value. In case of third conversion TDC serial is used as a key and a special structure **BmnTof1Parameters** (see app. B) is used as a value. This structure has INL data in it and an array of **BmnTof1Map2** structures, each of which stores plane ID, strip number and side of the strip. The index of this array is a TDC channel.

Such a way of storing mapping data provides a fast access to it (the computational complexity of searching the value by key in std::map container is $O(logN)$).

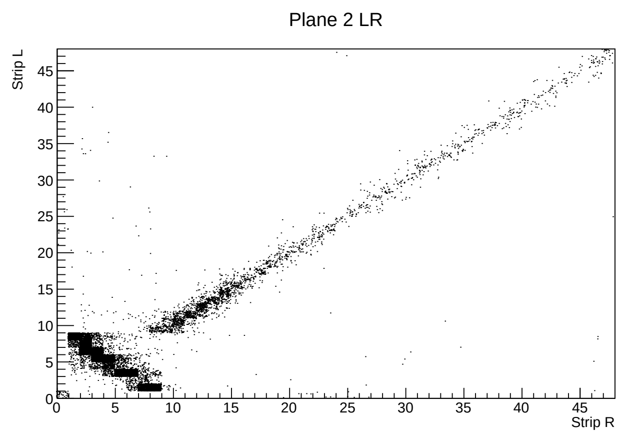
The hardest part of implementation of mapping is the determination of the third conversion's mapping table. Due to the equalization of the trace length on the front-end amplifier PCB purposes channels are already mixed up; moreover, some cables could be soldered incorrectly, which only increases the confusion. Unfortunately, the mapping table based on the PCB trace data and cable connection, provided by my supervisors, wasn't completely right because of some soldering and connection errors, and needed correction.

To correct this mapping table a couple of histograms were plotted:

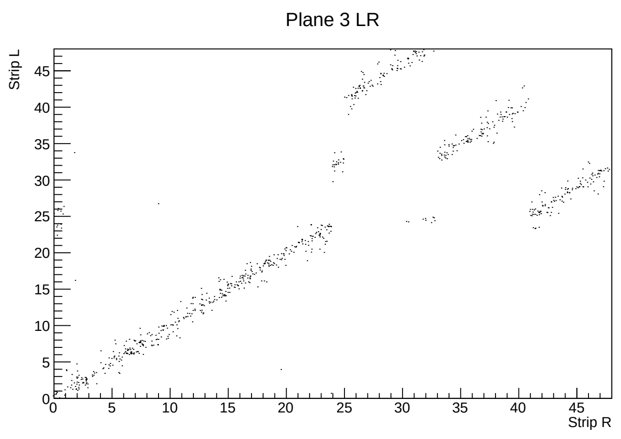
1. 3 2D histograms for each plane, in which X bin is the left-sided triggered strip number and Y bin is the right-sided triggered strip number (left-sided and right-sided strips with maximal amplitude (signal width) in the current event data were used to fill the histogram); (LR histograms hereinafter)

2. 3 2D histograms for each plane, in which the X bin is the left-sided triggered strip number and the Y bin is the right-sided triggered strip number of the closest by time left-right pair found in the event (T-LR histograms hereinafter);

3. 6 2D histograms for each plane and side of the strips, X bin is the strip number and Y bin is the TDC digit's leading time; (XT histograms hereinafter)

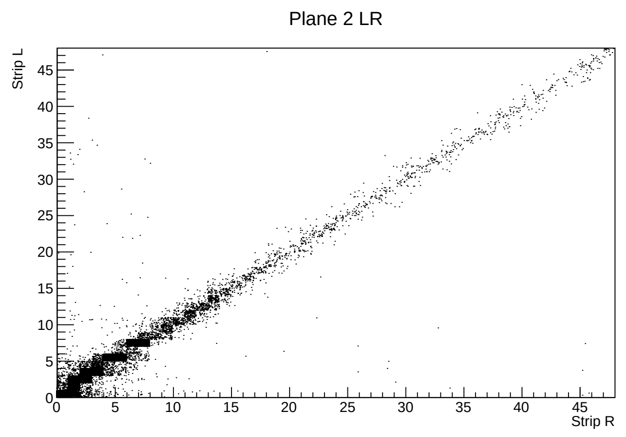


(a) 2nd plane LR histogram

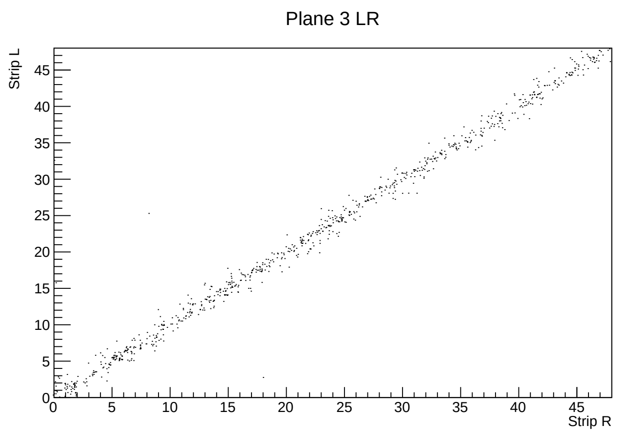


(b) 3rd plane LR histogram

Figure 4: LR histograms for 2nd and 3rd planes in case of uncorrected mapping



(a) 2nd plane LR histogram



(b) 3rd plane LR histogram

Figure 5: LR histograms for 2nd and 3rd planes in case of corrected mapping

In case of correct mapping table, the LR and T-LR histograms, obviously, should look like diagonals, because triggering the left side of the strip also triggers its right side. All of the six XT histograms should have no abrupt (high-deteriorative) zones in them and be the same for both sides of the same plane.

On fig. 4 the 2nd and 3rd plane LR histograms for the uncorrected mapping table are shown. It is obvious that the mapping has a couple of errors and needs correction. These errors were found and corrected using these histograms. Same histograms for the corrected mapping table are shown on the fig. 5. Applied corrections and the final mapping table could be found in the appendix A, full list of the built histograms is provided in the appendix C.

Fortunately, these errors will be eliminated in the future runs, so no complex mapping table confusion will occur in future.

A special ROOT macros, which uploads the mapping tables to the PostgreSQL database, has also been written during this work.

3.3 INL Correction

As said before, usage of the 24ns mode, in which the HPTDC chips operate in the TDC72VHL module, leads to large nonlinearity. To overcome this problem, this nonlinearity should be measured and then corrected using this collected data.

To define the INL (integral nonlinearity) term the DNL (differential nonlinearity) must be defined. The DNL for each of 1024 bins can be calculated using the equation (where $i = \overline{0, 1023}$ is the bin number, $j = \overline{0, 71}$ is the channel number and k represents the TDC72VHL module serial number)

$$DNL = \frac{W^r_{i,j,k} - W^t}{W_t}.$$

Here $W^r_{i,j,k}$ is the real bin i time width, and $W^t = 24ns/1024$ is the ideal bin width. The j and k indexes will be omitted hereinafter.

One can measure the DNL for each bin of each channel in the TDC72VHL module by applying a uniform-distributed in time LVDS pulse signals to the input of the TDC72VHL and calculating the number of events in each bin. Theoretically, the number of events in each bin should be proportional to the bin width, and, if the HPTDC chips were perfect (without any nonlinearity), number of events in each bin would be the same. In fact, the number of events varies from bin to bin, so does the bin width. Using such a procedure, one can find each bin's real width and, therefore, the differential nonlinearity.

The integral nonlinearity is defined as a sum over all of the DNL in "previous" bins:

$$INL_i = \sum_{x=0}^{i} DNL_x.$$

To convert the TDC Digit value v (bin id + number of full TDC cycles * total bin number in the channel) to real time in ns with the appliance of the nonlinearity correction, one can use an equation

$$t = \sum_{i=0}^{v\%1024} W_r^i + \left\lfloor \frac{v}{1024} \right\rfloor \times 24ns = \sum_{i=0}^{v\%1024} DNL_i \times W_t + W_t \times (v\%1024) + \left\lfloor \frac{v}{1024} \right\rfloor \times 24ns,$$

where $\lfloor x \rfloor$ is the floor operation, and $x\%y$ is the remainder of x divided by y.

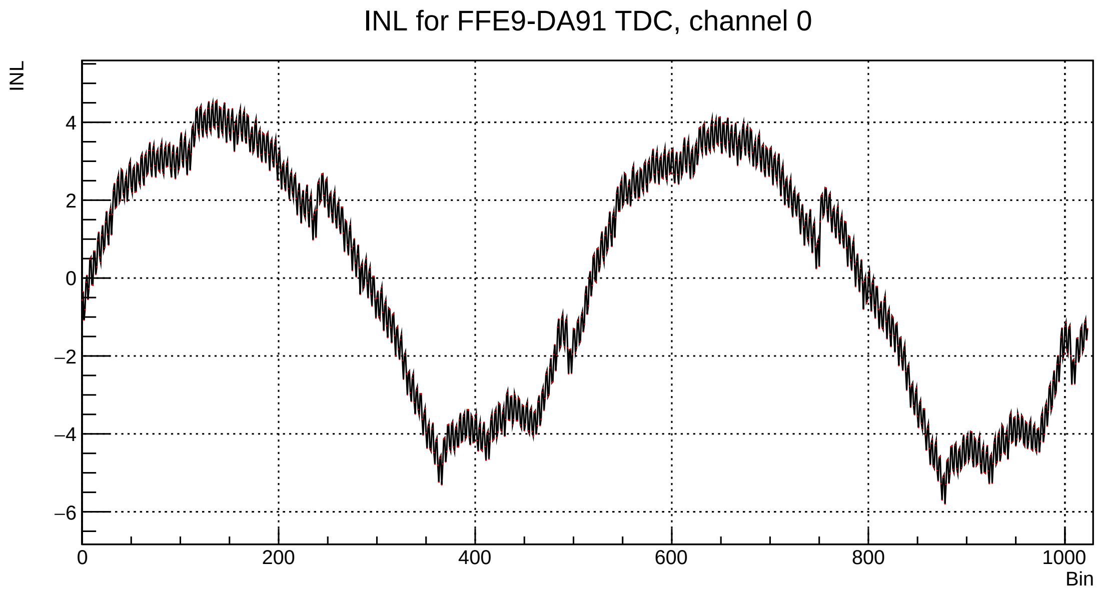


Figure 6: INL for the channel 0 of the FFE9-DA91 TDC72VHL module.

In terms of INL characteristics, this equation can be simplified to

$$t = \left(v + INL_{(v\%1024)}\right) \times \frac{24ns}{1024}. \tag{2}$$

Due to the comparative simplicity of this formula, INL characteristics are used to store the nonlinearity data and to apply the correction. An example of the INL characteristics for one channel is given on fig. 6.

The INL characteristics are stored in the database for each channel and TDC module separately as an array of double type values. The size of this array is equal to the number of bins – 1024. Before the conversion starts, these INL characteristics are loaded into 2D 72×1024 arrays of double type values inside each of the created during the mapping downloading process **BmnTof1Parameters** instance in the described in previous section std::map. The first dimension of each array represents the channel ID, while second dimension represents the bin ID.

3.4 Complete first conversion stage algorithm and determination of the signal amplitude

To find the signal amplitude we must find closest leading and trailing signals for the same strip and same side of it. To perform that, the **BmnTof1Parameters** class stores an array of temporary double values for each TDC channel **double BmnTof1Parameters::t[72]**. All of the temporary values are filled with “-1” before processing each event.

Also, all input TDC digits in the beginning of current event processing are sorted by time. While sorting the algorithm also "filters" the TDC data by ignoring all the TDC digits whose type is not valid (not 0x12 for TDC72VHL). This sorting is performed by inserting all filtered elements into the std::set container one by one.

Then, after the TDC digits are sorted and filtered, a loop over all of the sorted element in std::set container is performed. For each TDC digit the program performs these steps (we will define the TDC digit "time" as TDC Digit value converted to "real" time using eq. (2) and the corresponding INL characteristics):

1. Convert the HPTDC id and HPTDC channel to "common" TDC module channel using equation (1).

2. Using the TDC digit's fields for VHL crate serial and insertion slot of the triggered TDC as a key, find the TDC Serial in the mapping table described in section 3.2.

3. Using this TDC Serial as a key, find in the std::map the corresponding **BmnTof1Parameters** instance. From this instance we can extract mapping for each TDC channel (see section 3.2) and INL characteristics (see section 3.3), while also gaining access to the temporary storage in it **t**[**72**].

4. If the TDC digit is leading, just put the TDC digit time into the temporary storage **BmnTof1Parameters::t**[**72**] for the corresponding channel and TDC Serial. Go to step 6.

5. If the TDC digit is *not* leading, check the temporary storage for the corresponding channel and TDC Serial:

    (a) If the value stored in the temporary value array for the corresponding channel and TDC Serial is -1, it means that no leading TDC digit was found yet in this event for this strip. As the TDC Digits are sorted by time, it also means that the TDC digit we are currently on doesn't hold any information and we might as well just skip it. Go to step 6.

    (b) If the value stored in the temporary value array for the corresponding channel and TDC Serial is *not* -1, it means that we found a leading-trailing pair, where the leading time is stored in the temporary array value and the trailing time is the current TDC digit's time. Using mapping tables stored in the **BmnTof1Parameters** instance, find the corresponding plane ID, strip ID and side for this TDC Serial and channel. Fill the output array with these plane ID, strip ID, strip side, leading time (from the temporary storage) and signal amplitude in time equivalent ($t_{trailing} - t_{leading}$)

6. Go to the next TDC digit in the sorted list of them.

This loop continues until the program reaches the end of the input array of **BmnTDCDigit** instances.

This algorithm is implemented in the **BmnTof1Raw2Digit**'s method **FillEvent**. The **BmnRawDataDecoder** class itself handles the event loop and calls this method for each event, passing an array of all TDC digits in current event to the method using a TClonesArray of **BmnTDCDigit** instances and extracting a TClonesArray* of **BmnTof1Digit**.

The mapping table and INL characteristics can be loaded either from a local file using corresponding methods (see app. B) or from the PostgreSQL database. This downloading is performed only once per conversion process.

That way, on the output of the first conversion stage the Tof1Digit's amplitude is determined, and INL correction is applied, as well as the mapping table.

3.5 Complete second conversion stage algorithm

According to the aim of this work described in section 2.3, after the first stage conversion is complete, all that remains is to find different-sided pairs of **BmnTof1Digit** instances on the same strip and plane. This is performed by the **TofHitConvertor** (see app. B) class. Its ConvertDigitToHit method handles the event loop, passing a TClonesArray* of **BmnTof1Digit**s to the same classes' method FillEvent. The FillEvent method handles all the event processing and outputs a TClonesArray* of **myTof1Hit**. The algorithm implemented in the FillEvent method is described below and repeats for each event in the file.

1. Sort the **BmnTof1Digit**s first by plane, then by strip, and finally by side (right <= left) using the std::multiset container. That way in the sorted list we will have “clusters” of Tof1 digits with the same plane, strip and side.

2. Locate the current cluster of Tof1 digits with the same plane ID and strip (ignoring side sorting). This is done in $O(n)$ complexity by just finding the first digit after current, whose strip or plane is not equal to the current digit's strip and plane correspondingly.

3. Check the first Tof1 digit in the cluster. If it is on the left side of the strip, due to the sorting that means that there is no right-sided digits in this cluster (strip). Go to step 7.

4. Find the nearest by time left and right digits in the cluster using bruteforce approach (loop in a loop, $O(n^2)$ complexity). This can be done faster [9], but the number of digits in one strip in one event rarely exceeds 3, so almost no performance boost would occur.

5. Push a new **myTof1Hit** into the output array, using data from the found pair of Tof1 digits, filled with data (see below):

   (a) Hit time $= (t_{left} + t_{right})/2.0$;

   (b) Hit amplitude $= amp_{left} + amp_{right}$;

   (c) Hit position $= t_{left} - t_{right}$.

   (d) Strip and plane ID are filled from the right-sided digit in the pair

6. Remove the found pair from the sorted list. If the cluster is not empty, go to step 3.

7. Skip all the digits of the same strip and plane up to the next cluster (same strip and side as the digit found in step 3).

8. If the end of sorted list is not reached, go to step 2.

9. End of the sorted list was reached while trying to skip the cluster. That means that there is no more clusters with both left- and right-sided digits or nonempty ones, and, therefore, the conversion process for this event is complete.

During this conversion process the histograms described in section 3.2 are built (see app. C).

The **myTof1Hit** class stores the average time between left and right side. This increases the time resolution, as well as overcomes the position-time dependency. It also stores the position of interaction - difference between leading time on the left side and leading time on the right side, as well as the hit amplitude - a sum of the amplitudes on the left and right side. Obviously, it also stores the plane ID and strip number.

As said before, the **TofHitConverter** class performs this conversion on each event in the input file. It outputs the events to a different file containing a TTree filled with TClonesArray's of **myTof1Hit** along with the debug histograms.

4 Conclusion

In this work a data preprocessing complex for ToF-400 system in the BM@N experiment was developed. The complex features:

- INL Correction;
- Mapping;
- Signal amplitude measurement, combining process of the leading and trailing signals;
- Combining process of TDC digits on different sides of one strip;
- Measurement of interaction position along the strip.

The complex is built around the BMNRoot framework and is easily expandable and editable, as the developed complex is easily configurable. It preloads all the settings from the PostgreSQL database; a couple of macros for data upload into the database were also written.

The developed system was tested on the test run data and may be used in all upcoming runs.

5 References

[1] Ablyazimov T. O. et al. (BM@N Collaboration) // Conceptual Design Report of BM@N. http://nica.jinr.ru/files/BM@N/BMN_CDR.pdf

[2] Afanasiev S. V. et al. (BM@N Collaboration) // BM@N Project Prologation for 2017-2021.

[3] ToF-400 status for project prolongation, March 2016. http://bmnshift.jinr.ru/wiki/lib/exe/fetch.php?media=tof_wall400_for_htc.docx

[4] Anghinolfi F. et al. // NINO: an ultra-fast and low-power front-end amplifier/discriminator ASIC designed for the multigap resistive plate chamber, Nucl. Instrum. Meth. A 533 (2004) 183.

[5] Christiansen J. // High performance time to digital converter (HPTDC), Digital Microelec. Group, CERN, HPTDC manual version 2.2 for HPTDC version 1.3.

[6] NICA / Bmnroot – Simulation and Analysis Framework for NICA/BM@N Detectors. https://git.jinr.ru/nica/bmnroot

[7] Root - Cern. An object oriented framework for large scale data analysis. https://root.cern.ch/

[8] FairRoot. https://fairroot.gsi.de/

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein // Introduction to algorithms, third edition.; MIT Press, 2009. p. 1039–1043 of section 33.4: Finding the closest pair of points.

Appendices

A Corrected mapping

Originally provided mapping tables:

Crate serial	Slot	TDC Serial	TDC Channels	Plane	Side	Strips
01690454	4	FFE9DA91	48-71	1	Left	0-23
01690454	4	FFE9DA91	24-47	1	Left	24-47
01690454	4	FFE9DA91	0-23	1	Right	0-23
01690454	5	076CCF62	48-71	1	Right	24-47
01690454	5	076CCF62	24-47	2	Left	0-23
01690454	5	076CCF62	0-23	2	Left	24-47
01690454	6	076CAD53	48-71	2	Right	0-23
01690454	6	076CAD53	24-47	2	Right	24-47
01690454	6	076CAD53	0-23	3	Left	0-23
01690454	7	FFE9F5CB	48-71	3	Left	24-47
01690454	7	FFE9F5CB	24-47	3	Right	0-23
01690454	7	FFE9F5CB	0-23	3	Right	24-47

Table 1: Uncorrected mapping. All the serials are written in hexademical

Side	Table												
Left	Channel	0	1	2	3	4	5	6	7	8	9	10	11
	Strip	4	1	10	7	5	2	11	8	6	3	12	9
	Channel	12	13	14	15	16	17	18	19	20	21	22	23
	Strip	19	22	13	16	20	23	14	17	21	0	15	18
Right	Channel	0	1	2	3	4	5	6	7	8	9	10	11
	Strip	21	0	15	18	20	23	14	17	19	22	13	16
	Channel	12	13	14	15	16	17	18	19	20	21	22	23
	Strip	6	3	12	9	5	2	11	8	4	1	10	7

Table 2: Mapping for each of the 24 channels / 24 strips pair (each line of the tab. 1)

Corrections that should be made after applying the mapping based on the tables 1 and 2:

1. Cyclically shift all the strips towards the zero strip in the second table (Strip 1 becomes strip 0, strip 0 becomes 23, strip 2 becomes strip 1 and so on);
2. Swap the 24-32 and 39-47 strips in the right side of plane 3 (strip 24 swaps with strip 39, strip 25 swaps with strip 40 and so on);
3. Reverse the order of 0-7 strips in the left side of plane 2 (strip 0 swaps with strip 7, strip 1 swaps with strip 6 and so on).

B Classes and structures description

All listings are provided in a shortened form.

B.1 BmnTDCDigit

Listing 1: BmnTDCDigit.h

```
class BmnTDCDigit : public TObject {
public:
  // Default constructor
  BmnTDCDigit();

  // Constructor to use
  BmnTDCDigit(UInt_t iSerial, UChar_t iType, UChar_t iSlot, Bool_t iLeading,
   UChar_t iChannel, UChar_t iHptdcId, UInt_t iValue);

  //Getters
  UInt_t GetSerial() const {return fSerial;}
  UChar_t GetType() const {return fType;}
  UChar_t GetSlot() const {return fSlot;}
  Bool_t GetLeading() const {return fLeading;}
  UChar_t GetChannel() const {return fChannel;}
  UInt_t GetValue() const {return fValue;}
  UChar_t GetHptdcId() const {return fHptdcId;}

  //Destructor
  virtual ~BmnTDCDigit();
  private:
  UInt_t fSerial; //VHL crate Serial
  UChar_t fType; //TDC type
  UChar_t fSlot; //VHL Slot
  Bool_t fLeading; //True for leading digits, False for trailing
  UChar_t fChannel; //HPTDC Channel
  UChar_t fHptdcId; //HPTDC Chip ID
  UInt_t fValue; //Value
};
```

B.2 BmnTof1Digit

Listing 2: BmnTof1Digit.h

```
class BmnTof1Digit : public TObject
{
  public:
    //Default, normal and "copying" constructors
    BmnTof1Digit();
    BmnTof1Digit(Short_t plane, Short_t strip, Short_t side, Float_t t, Float_t a
   );
    BmnTof1Digit(const BmnTof1Digit*, Float_t t, Float_t a);
    //Destructor
    virtual ~BmnTof1Digit();

    //Getters
    Short_t GetPlane() const { return fPlane; }
    Short_t GetStrip() const { return fStrip; }
    Short_t GetSide()  const { return fSide;  }
    Float_t GetAmplitude() const { return fAmplitude; }
    Float_t GetTime()  const { return fTime;  }

    //Setters
    void SetPlane(Short_t v) { fPlane = v; }
    void SetStrip(Short_t v) { fStrip = v; }
    void SetSide (Short_t v)  { fSide = v; }
    void SetAmplitude(Float_t v){ fAmplitude = v; }
    void SetTime (Float_t v){ fTime = v; }

    //Prints the information stored in this Tof1Digit
    void print(const char* comment = nullptr, std::ostream& os = std::cout)const;
  private:
    Float_t fAmplitude; //Amplitude
    Float_t fTime;     //Leading time
    Short_t fPlane;    //Plane ID
    Short_t fStrip;    //Strip number
    Short_t fSide;     //Side (1 for Left, 0 for right)
};
```

B.3 BmnTof1Raw2Digit

B.4 BmnTof1TDCParameters

B.5 BmnTof1Map2

Listing 3: BmnTof1Raw2Digit.h

```
//Side of the strip is stored as a bool variable
#define TOF1_LEFT true
#define TOF1_RIGHT false

//Parameters
#define TOF1_CHANNEL_NUMBER 72 //Number of channels
#define TOF1_BIN_NUMBER 1024 //Number of bins in a channel
#define TOF1_MAX_TIME (24.) //In ns
#define TOF1_TDC_TYPE (0x12) //TDC72VHL type

//A simple comparsion functional class to compare the TDCDigits
//Sorts them by time
struct _Tof1TDCCompare {
  bool operator()(const BmnTDCDigit& a, const BmnTDCDigit& b);
};

//Map element
struct BmnTof1Map2 {
  Short_t plane;
  Short_t strip;
  Bool_t side;
  //Constructors
  BmnTof1Map2(Short_t, Short_t, Bool_t);
  BmnTof1Map2();
};

//TDC parameters
struct BmnTof1TDCParameters {
  double INL[TOF1_CHANNEL_NUMBER][TOF1_BIN_NUMBER]; //INL
  BmnTof1Map2 ChannelMap[TOF1_CHANNEL_NUMBER]; //A BmnTof1Map2 for every channel
  double t[TOF1_CHANNEL_NUMBER]; //Temporary value
  BmnTof1TDCParameters(); //Simple constructor
};

class BmnTof1Raw2Digit {
  public:
    //BmnTof1Raw2Digit main constructor
    BmnTof1Raw2Digit();
    //Another constructor, calls setRun(...)
    BmnTof1Raw2Digit(int nPeriod, int nRun);
    //Destructor
    ~BmnTof1Raw2Digit();

    //Loads mapping and INL from the DB for run #nRun in period #nPeriod
    void setRun(int nPerion, int nRun);

    //Load mapping from two files
    void setMapFromFile(std::string placementMapFile, std::string mapFile);
    //Save the mapping to two files
    void saveMapToFile(std::string placementMapFile, std::string mapFile);

    //Loads INL from an INI file
    void setINLFromFile(std::string INLFile);
    //Saves INL for TDCSerial to an INI file
    void saveINLToFile(std::string INLFile, unsigned int TDCSerial);

    //Main conversion function
    void FillEvent(TClonesArray *data, TClonesArray *tof1digit);

    //Returns TDC Channel
    static UShort_t ToGlobalChannel(UChar_t HptdcId, UChar_t channel);
  private:
    //BmnTof1Raw2Digit init function (called in BmnTof1Raw2Digit constructors)
    void init();
    //RunIndex and PeriodIndex
    int RunIndex, PeriodIndex;

    //Stores the placement map
    std::map<std::pair<UInt_t, UChar_t>, UInt_t> PlacementMap;

    //Stores the loaded main mapping
    std::map<UInt_t, BmnTof1TDCParameters> TDCMap;

    //Inserts a value in the placement map
    void plmap_insert(UInt_t Serial, UChar_t Slot, UInt_t TDCSerial);
};
```

B.6 myTof1Hit

Listing 4: myTof1Hit.h

```
class myTof1Hit :public TObject {
  public:
    //Constructors
    myTof1Hit(UInt_t plane, UInt_t strip, Double_t time, Double_t amp, Double_t
   pos);
    myTof1Hit();

    //Getters
    UInt_t GetPlane();
    UInt_t GetStrip();
    Double_t GetTime();
    Double_t GetAmplitude();
    Double_t GetPosition();
  private:
    UInt_t iPlane;    // Plane ID
    UInt_t iStrip;    // Strip number
    Double_t fTime;   // Mean time
    Double_t fAmplitude;  // Hit amplitude
    Double_t fPosition; // Hit position
};
```

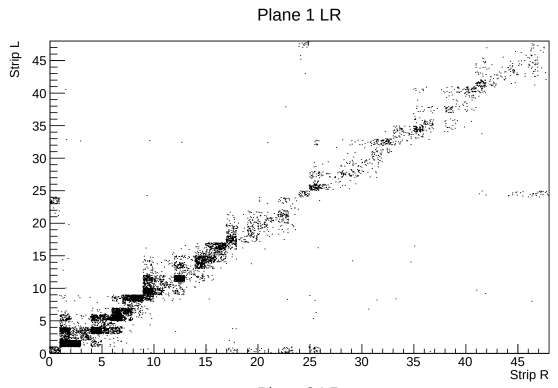
B.7 TofHitConverter

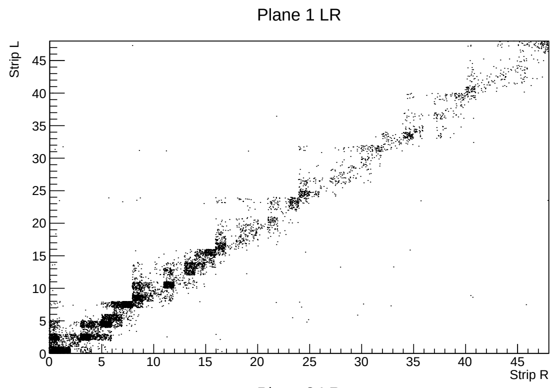
Listing 5: TofHitConverter.h

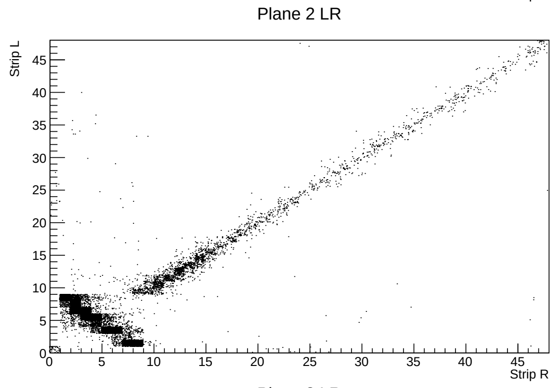
```
//Comparsion functional class
class _Tof1DigitPointerCompare {
  public:
    bool operator() (BmnTof1Digit* a, BmnTof1Digit* b);
};

class TofHitConverter {
  public:
    //Constructors
    TofHitConverter();
    TofHitConverter(std::string inFile, std::string outFile);
    void init();

    //Setters
    void SetDigitFile(std::string inFile);
    void SetHitFile(std::string outFile);

    //Main functions
    //Convert the DigitFile to HitFile
    void ConvertDigitToHit(Int_t lookat = -1);

    //Convert one event
    void FillEvent(TClonesArray* digit, TClonesArray* hit, bool debug);
  private:
    //Histos
    TH2D* h_xy[3];     //LR Histograms
    TH2D* h_xTy[3];    //T-LR Histograms
    TH2D* h_xt[6];     //XT Histograms

    std::string digitFilename;  //Input filename
    std::string hitFilename;  //Output filename

    TFile* digitFile; //Input TFile
    TFile* hitFile;   //Output TFile
    TTree* digitTree; //Input TTree
    TTree* hitTree;   //Output TTree

    //Creates a hit from two paired TDC Digits
    void createHit(TObject* where, const BmnTof1Digit* left, const BmnTof1Digit*
   right);
};
```

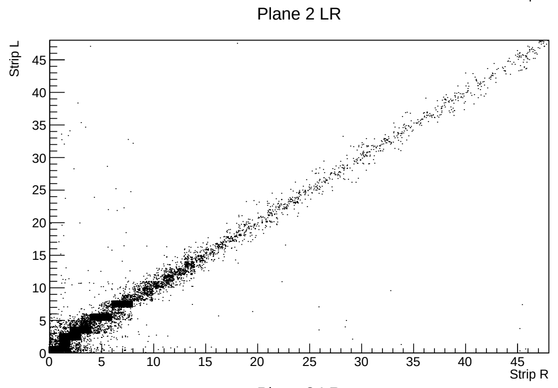
C Complete list of built debug histograms

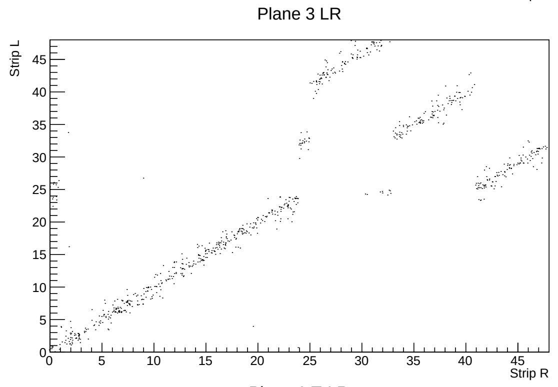
On the left histograms for the uncorrected mapping are shown, on the right - same histograms after correction. Histograms are labeled above them.

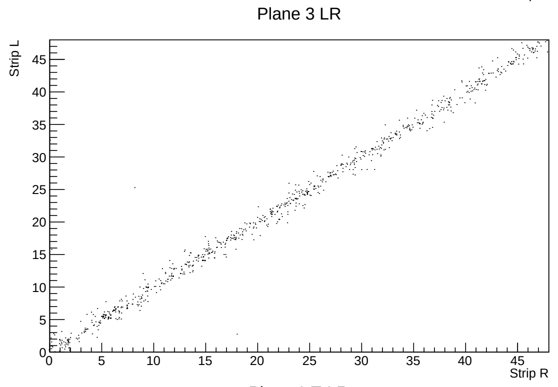


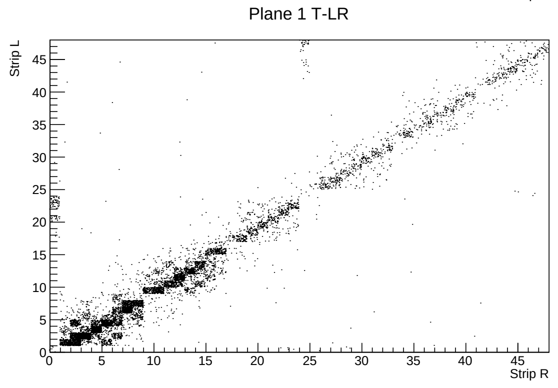


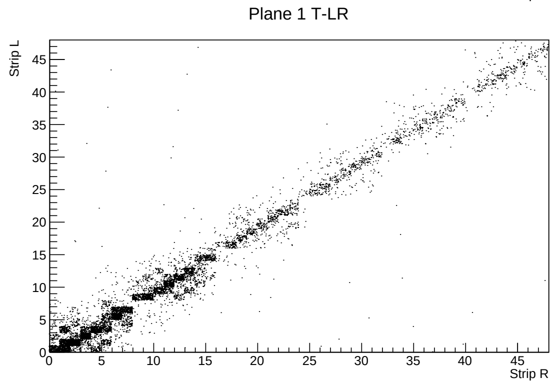


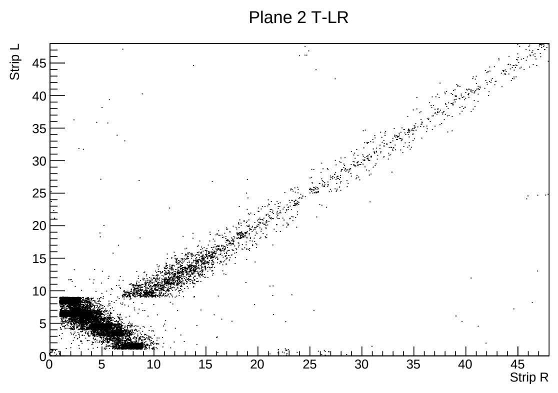


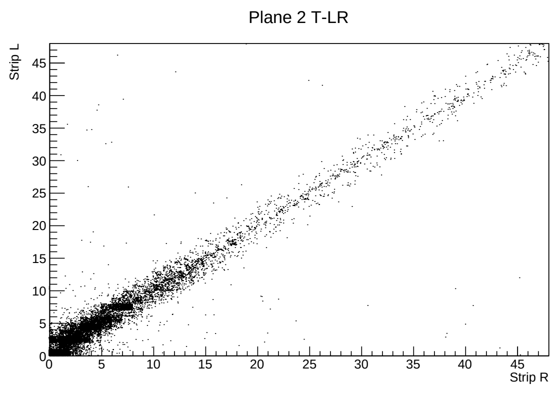


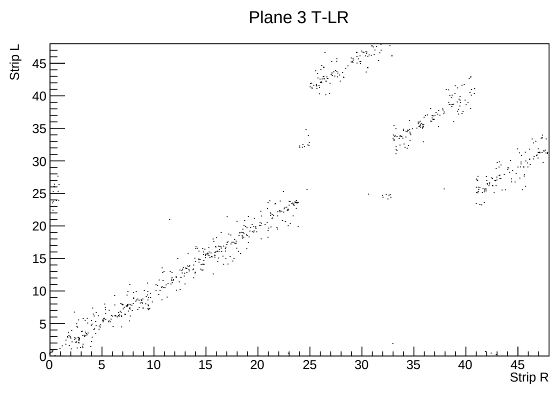


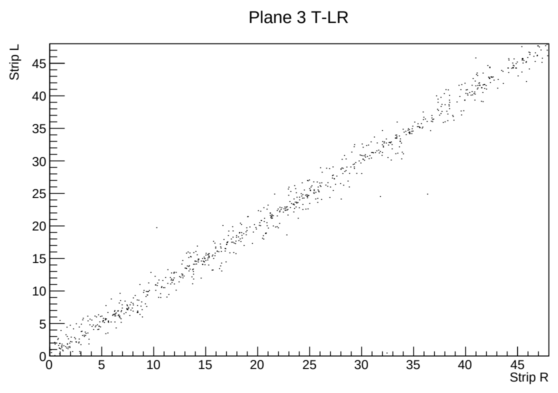


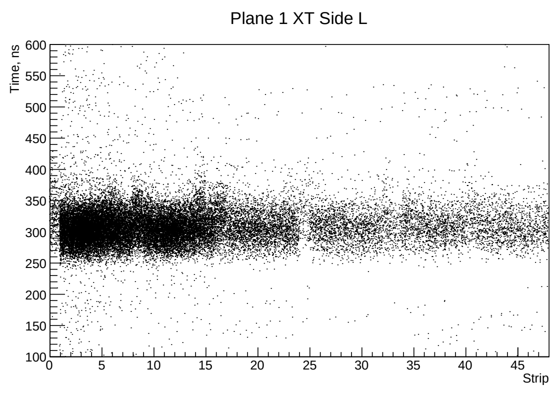


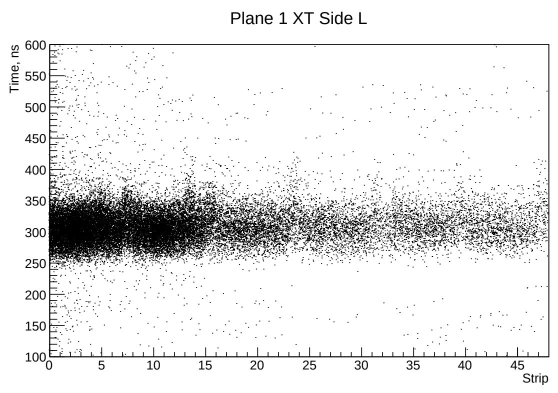

Plane 2 T-LR
Strip L
Strip R

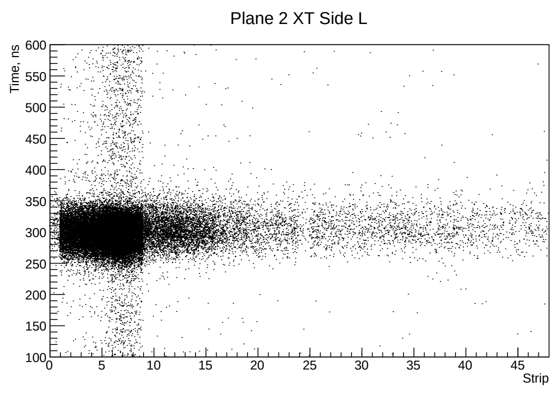

Plane 2 T-LR
Strip L
Strip R

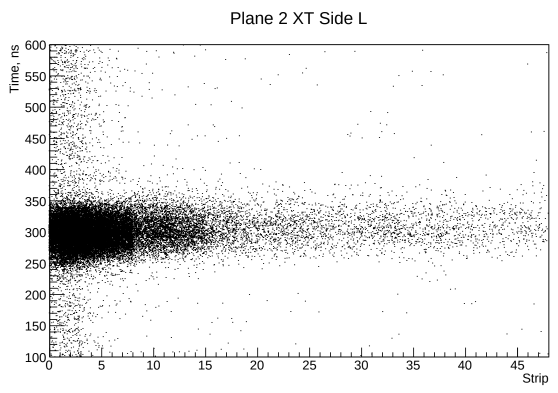

Plane 3 T-LR
Strip L
Strip R

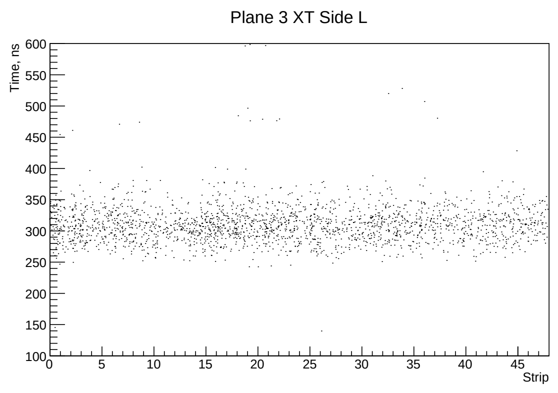

Plane 3 T-LR
Strip L
Strip R

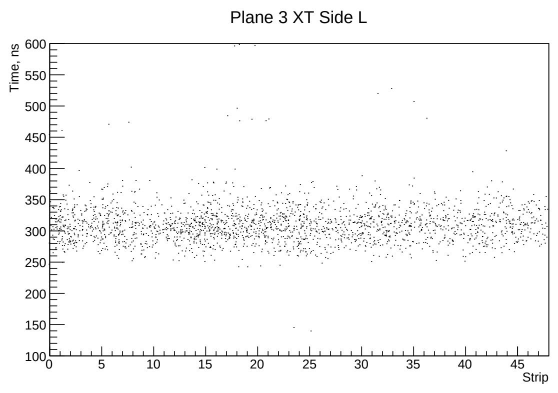

Plane 1 XT Side L
Time, ns
Strip

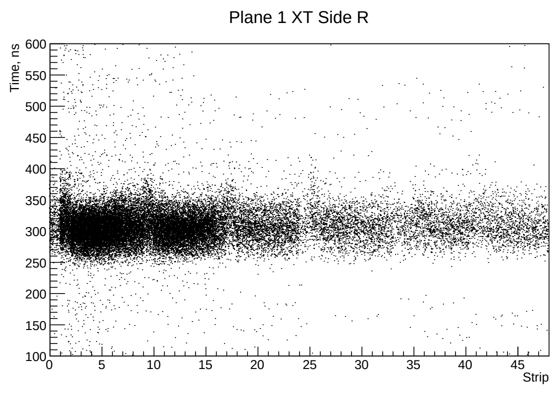

Plane 1 XT Side L
Time, ns
Strip

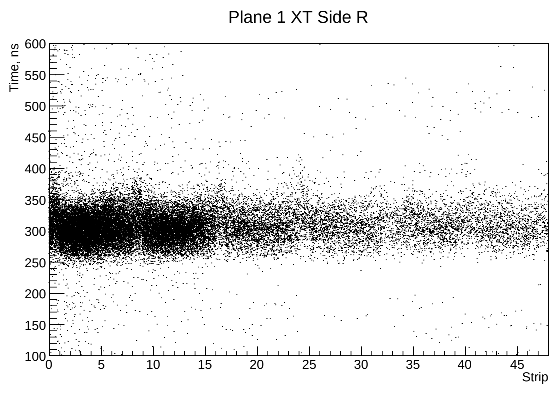

Plane 2 XT Side L
Time, ns
Strip

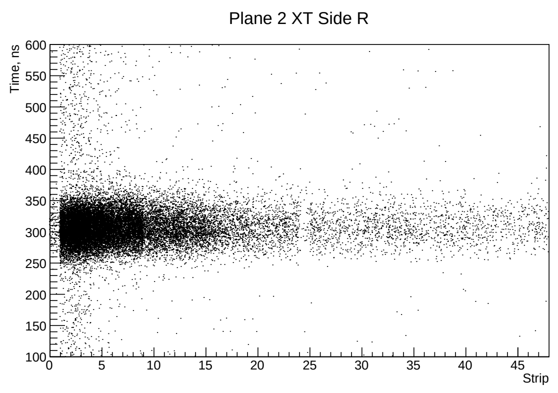

Plane 2 XT Side L
Time, ns
Strip

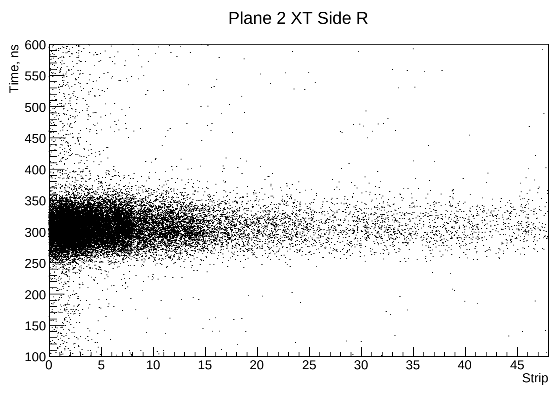

Plane 3 XT Side L
Time, ns
Strip

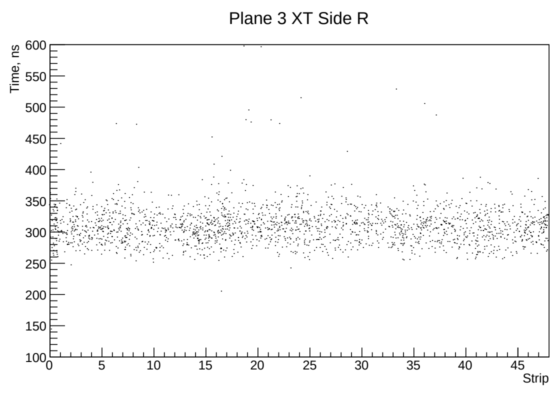

Plane 3 XT Side L
Time, ns
Strip

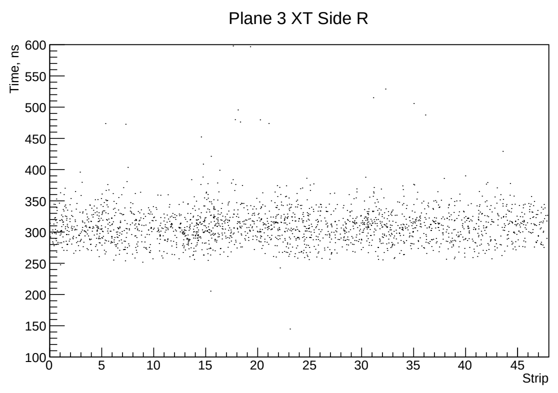

Plane 1 XT Side R
Time, ns
Strip


Plane 1 XT Side R
Time, ns
Strip


Plane 2 XT Side R
Time, ns
Strip


Plane 2 XT Side R
Time, ns
Strip


Plane 3 XT Side R
Time, ns
Strip


Plane 3 XT Side R
Time, ns
Strip