CLAS-NOTE 2003-004
CVS: docs/clas_notes/new_tagger_calib
Original: March 17, 2003
Version: 1.0

The New Tagger Calibration Program

Ji Li

1 Introduction

Timing is the cornerstone of particle identification and any other off-line data analysis that follow. Its importance can never be over-emphasized. For photon experiments, timing calibration and reconstruction are more difficult than for electron experiments in the sense that the accelerator RF time is not directly used. The time of the beam photon is measured by the CLAS Bremsstrahlung tagging system(Tagger) and then corrected for the phase difference between the tagger time and the RF time. The alignment of those two times is the most important aspect of the tagger calibration.

In this note, I will briefly describe the procedure for the tagger calibration. For more details on tagger reconstruction and calibration, please refer to CLAS_NOTE 1999-04. Detailed information about implementation and running of the calibration program `photonTcal` is given.

2 Tagger calibration procedure

Four sets of constants are being used in the tagger reconstruction software, as shown in table 1. All the constants are kept in TAG_CALIB.map under $CLAS_PARMS/Maps directory[1]. Another set of constants, which are used in RF time reconstruction, are kept in RF_OFFSETS.map under the same directory.

The goal of the tagger calibration is to adjust all these constants, so that the beam photon timing can be reconstructed correctly, and eventually allows substitution of the tagger time with the accelerator RF time on an event by event basis. Therefore the calibration consists of 5 major steps each corresponds to one of the 5 sets of constants. The formalism used in these steps is well documented in CLAS_NOTE 1999-04. I will not go through them here and strongly recommend any serious tagger calibrator to read that note. Also, there are dependencies among these steps, thus they have to be carried out in the following order:

1. RF calibration
   There are two things regarding the RF calibration that the calibrators should be aware of,

[1] *Map* refers to both the calibration map and the database.

Number of constants	Map entry (**subsystem**/item)
2 x 61 constants for L/R T-counter TDCs, 1 constant for E-counter TDCs	**tag_t**/slope_left/right
2 x 61 constants for L/R T-counter TDCs, 384 constants for E-counter TDCs	**tag_e**/dt; **tag_t**/dt_left/right
121 constants, one for each T-bin	**tag_t**/ci
1 constant	**tag2tof**/value
4 x 6 + 1 constants for 4 regions of RF time	**F(1-4)**/low/high/p(1-4); **offset**/value

Table 1: Constants used in the tagger timing reconstruction.

(a) The RF signal from the accelerator is prescaled by a factor of 40 for photon runs, then divided into two identical signals. The choice of RF1 or RF2 is arbitrary. It would have been ideal to use the average of the two signals in computing the RF time. However, study showed that using the average did not make much difference.[2]

(b) Due to the physical property of the RF TDCs, the reconstructed RF time is divided into 4 regions. This is why in RF_OFFSETS.map we have four regions. The 'edge-finding' for those regions can not be easily automated. The way I did it was I zeroed out(meaning setting all the constants to zero) all the constants in the map for all the four regions, then processed some events with a small program that reconstructs RF time and tagger time. Then I plotted *Ttag-Trf vs. Trf*. I am not including that program here, because the same thing can be done with `photon_mon`, histograms *h72*, *h73* and *h74*. Fortunately, this need not be done very often. To check if this re-division is needed, one can proceed with the old constants and take a look at those three histograms. If the edges are improperly set, you will see obvious wiggles around the edges.

2. TDC base peak positions
The T-counter TDCs work in common start mode. The CLAS trigger sets the start signal and the electron hit sets the stop signal. Since the tagger signal itself is also part of the CLAS trigger, the T-counter TDCs are working in a so-called 'self timed' mode. This means the same signal starts and stops the TDC at different times. Because the cable lengths and the electronics delays are constant, the raw TDC spectrum shows a sharp peak. By convention, this amount of time is subtracted from the TDC value in calculating the tagger time, so the absolute value of the tagger time is close to zero. The E-counter TDCs are also working in essentially the same way even though they are in common stop mode. This step of the calibration is to find the centroid of the peak in the TDC spectra for each individual TDC.

3. TDC slopes
The E-counter TDC slopes are fixed at $500 ps/channel$. The T-counter TDC slopes are close to $50 ps/channel$. However, they vary slightly from counter to

[2]Personal communication with Dr. Elton Smith.

counter. The goal of this calibration is to find the correct slope for each T-counter TDC, both left and right. The slopes for the left/right TDCs on the same T-counter also need to be balanced so that the position of the hit on the hodoscope does not bias the timing.

4. Alignment of the tagger time and the RF time(Ci)
   As mentioned earlier, there is a phase difference between the tagger time and the accelerator RF time. This phase difference is represented by the set of so-called 'Ci' constants. Upon correctly determining these constants, one can substitute the tagger time with the more accurate RF time.

5. Alignment to CLAS(tag2tof)
   This constant represents the fixed relative delay between the tagger and the time-of-flight counter.

3 Implementation and compilation of the program

`photonTcal` is written in C++ by calling standard ROOT libraries. `fpack` is used for BOS I/O. All the histogram filling and fitting are done within the program. This program was originally developed on a RedHat Linux7.1 platform, with g++-2.96 and ROOT 3.01/05. It also has been compiled successfully on RedHat Linux6.2, with egcs 2.91.66 and ROOT 3.01/06. A few words are in order on compilation. People have been debating that ROOT based programs tend to be more unstable and harder to compile on different platforms. One realizes though, this is the nature of any dynamically linked program. There should be no problem with compilation if one is careful enough with the shared libraries. Just keep this in mind, you have to compile your ROOT libraries and `photonTcal` with the same compiler and libraries, packlib say.

Before compile the program, some environment variables must be setup correctly[3]:

$ROOTSYS This is THE most important variable concerning ROOT. It should be set to your ROOT directory.

$LD_LIBRARY_PATH Shared library path, should include your $ROOTSYS/lib.

$CERN_ROOT CERN library path, make sure it points to the same location that you built your ROOT libraries with.

Here is an example how I setup those variables when I built `photonTcal` on ifarml1(tcsh).

```
ifarml1>setenv ROOTSYS /u/group/clasdev/root
ifarml1>setenv CERN_ROOT /apps/cernlib/pc_linux/99
ifarml1>setenv LD_LIBRARY_PATH $ROOTSYS/lib:/apps/egcs/egcs-1.1.1-shared/lib
```

Once you setup these variables, you can check out `photonTcal` via `CVS` and compile it. At command line, type

```
ifarml1>cvs co /packages/utilities/photonTcal
```

[3]I take it here that all the CLAS related variables have already been setup properly. Otherwise please refer to the Hall-B software FAQ.

```
ifarml1>cd packages/utilities/photonTcal
ifarml1>make lib
ifarml1>make photonTcal_main
```

If everything went well, you should see `photonTcal_main` under current directory.

4 Running photonTcal

`photonTcal` works on raw BOS data. The behavior of `photonTcal` is controlled by command line flags. Type

```
ifarml1>photonTcal_main -h
```

you can see the help page that comes with the program.

```
photonTcal_main
   options are:
   -o   output ROOT file name, .root extension recommended.
   -t#  trigger mask.
   -M#  process maximum # of events.
   -S#  scale number of events for creating control histogram(default=1).
   -i   quiet mode (no counter).
   -s#  use this number for bank sector(default = 0)
   -P#  Bitwise Process flags, as defined below:
        0x1     Rough adjustment of E-T peak position.
        0x2     T-counter slope calibration.
        0x4     E-T peak position calibration.
        0x8     RF timing calibration.
        0x10    Ci calibration.
        0x20    TOF timing calibration.
        0x40    Creat control histograms.
   -R#  specify reference detector(default = ST). TBI=To Be Implemented
        0       ST
        1       TOF
        2       TAC(TBI)
        3       PS(TBI)
        4       EC(TBI)
   -h   print the above
   For more information, please see help.html under photonTcal directory.
```

For a normal calibration run[4], I would like to recommend the following sequence of running `photonTcal`

- Calibrate the RF first
  `ifarml1>photonTcal_main -P0x49 -orun12345.root clas_012345.A00`
- Calibrate the tagger
  `ifarml1>photonTcal_main -P0x57 -orun12345.root clas_012345.A00`

[4]From our experience, a special run taken with low photon flux, $5nA$ on a 10^{-4} radiator for instance, is desirable for the tagger calibration. A couple of million events should suffice the job.

Usually 300,000 events is the bottom line for calibrating a normal run, 500,000 events will give you a better fitting result. The dependencies of each calibration step are described in Tagger calibration procedure. In general, earlier step will affect later step(s). The dependencies have been hard-coded in the program. Which means you do not have to worry about things like missing a step. For instance, if you choose to run with -P0x2, the program will also do 0x5c for you. The reason that we setup this process flag is because some of the constants stay stable during a certain period of time, thus you do not have to recalibrate them very often. Nevertheless, `photonTcal` runs pretty fast, it would not burden you too much if you do the whole thing. My suggestion is to run with the flag -P0x57, this means do everything except RF calibration which is believed to be stable. If the running condition did not change, you should not need to calibrate every run. One way to determine the necessity of calibration for a given run is to run `photonTcal` with -P0x41 flag. This will create the control histograms with the old constants. Check the histograms and see if any calibration needs to be done on that run. Another commonly used program serving the same purpose is `photon_mon`. This is the standard cooking monitoring routine located at `packages/utilities/photon_mon`.

`photonTcal` writes out one ROOT file and several ASCII files. The ROOT file contains all the histograms that `photonTcal` uses to do the fits as well as control histograms for calibration quality checking. The ASCII files contain all the new constants that will go into the calibration map. The base-names of the ASCII files are fixed, while the extension is the run number. To look at the histograms, one needs to open a ROOT interactive session. If your $ROOTSYS and $PATH are setup correctly, typing `root` at command line will do it. Once you are in ROOT, you have to open the ROOT file created by `photonTcal` first, followed by opening ROOT Object Browser[5], see figure 1.

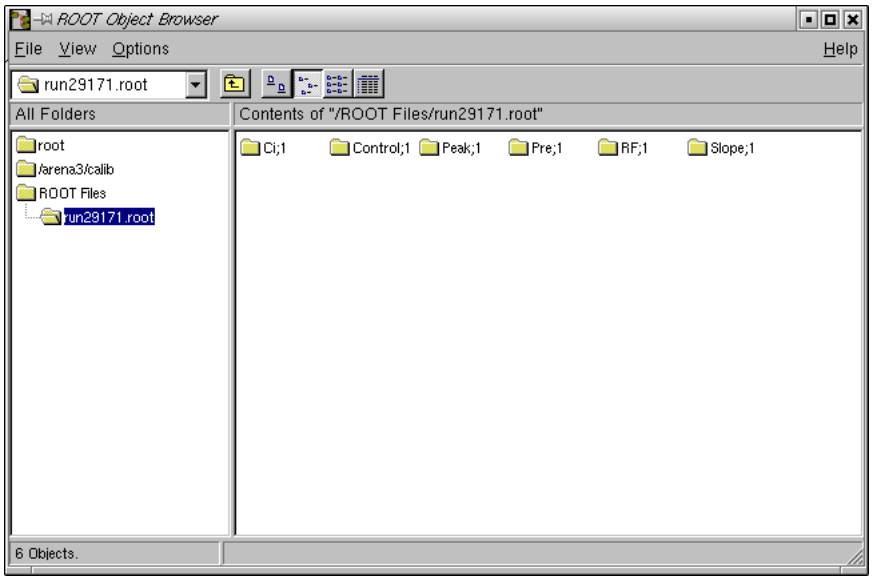

Figure 1: Illustration of a ROOT object viewer.

Among those 6 folders, Control is of the most importance. It contains all the quality control histograms. The other folders contain all the intermediate histograms

[5]Please find instructions for ROOT at *http://root.cern.ch*

used by `photonTcal` to do the fittings for different calibration steps, as indicated by the names of the folders. They are designed for expert use. How to interpret the control histograms is described in section 5. As stated above, the new calibration constants are stored in several ASCII files:

tagposEpeak.new.12345 E-counter TDC peak position, 2 columns x 384 rows

tagposTpeak.new.12345 T-counter TDC peak position, 3 columns x 61 rows

tagTDCCal.new.12345 T-counter TDC slopes, 3 columns x 61 rows

tagCalCi.new.12345 Ci, 2 columns x 121 rows

A shell script `putConst.sh` is included to facilitate putting the new constants into TAG_CALIB.map. This script takes 2 arguments, the first one is the *run number* and the second one is *$CLAS_PARMS*.

5 Calibration quality control

The quality of the calibration is controlled by checking several histograms. Double-click on the Control folder, you will see the histograms contained in that folder as shown in figure 2.

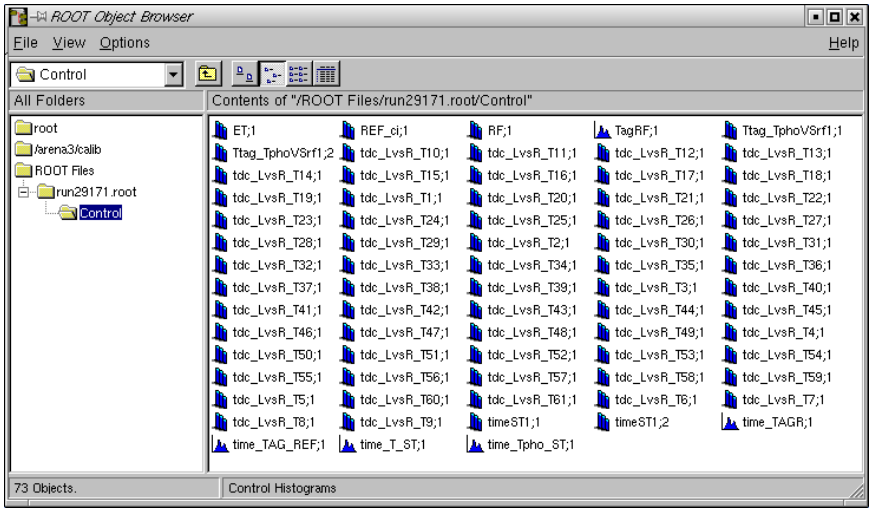

Figure 2: Contents of the control histogram folder.

The T-counter Left/Right TDC slope balance is controlled by 61 two-dimensional histograms, labled tdc_LvsR_T#, where # is the paddle number. The Y-axis in these plots is (tRight - tLeft)/2, while the X-axis is (tRight + tLeft)/2. If the L/R TDCs are well balanced for each T-counter, one should see a horizontal band around zero, as shown in the letf plot below. In the right plot in figure 3, one clearly sees that left TDC(or maybe PMT) is dead on that counter.

The E-T time coincidence is reflected in histogram ET. In this historgram, the Y-axis is the difference between E-counter time and T-counter time, X-axis is E-bin ID. If the calibration is good, we expect to see a horizontal bar around zero, which means the E-couters and T-counters are in time and there is no systematic dependency in

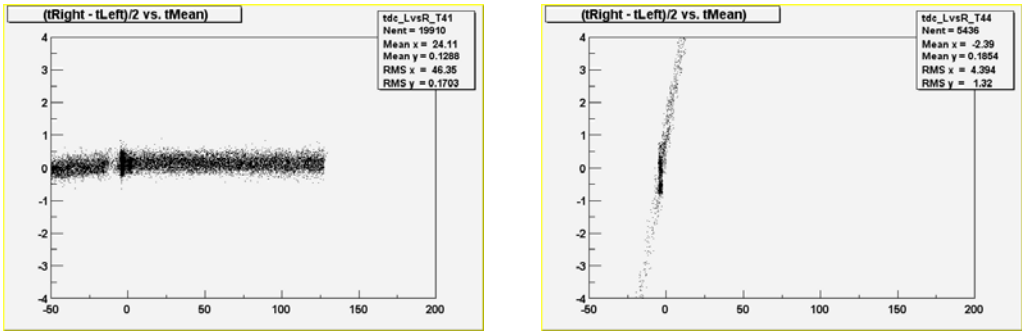

Figure 3: T-counter left/right TDC balance.

E-bin. One has to bear in mind that there is a hard-cut in the tagger reconstruction package, which is +/-10 ns for this time window at the time this document is written. As long as the hit falls within this window, the time of the beam photon is determined solely by T-counter time and eventually by the RF time.

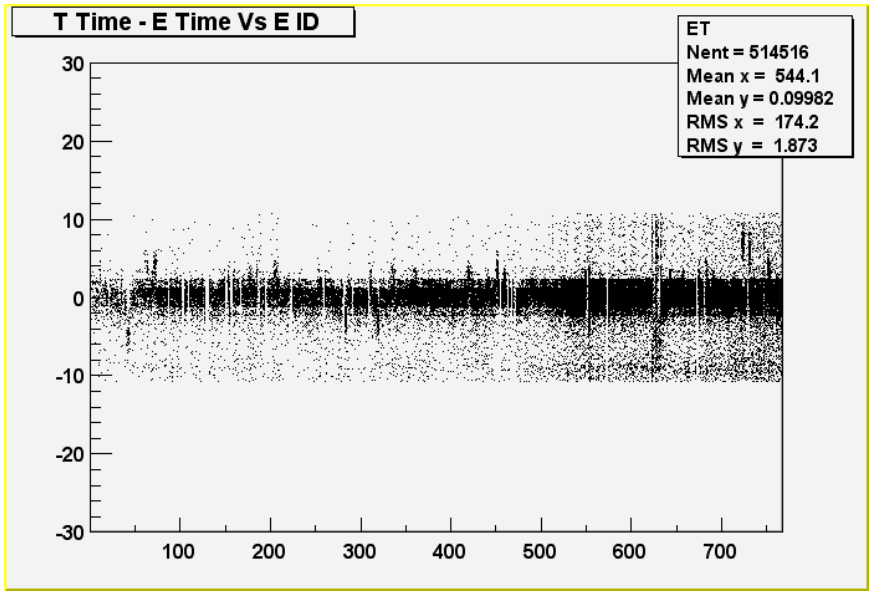

Figure 4: E/T-counter timing alignment.

The RF offset calibration is checked by looking at histogram Ttag-TphoVSrf1. Y-axis is Ttag - Tpho, X-axis is RF1. For a good calibration, one should see no slopes or offsets in any of the 4 regions.

The alignment between tagger time and RF time is checked with three histograms, as listed in table 2.

Histograms RF and TagRF show how good the *PHASE SHIFT* between tagger time and RF time is determined by the Ci constants. RF also shows that this alignment should not depend on T-bin after a good calibration. The sigma of TagRF reflects the intrinsic time resolution of the T-counters.

However, these two plots are not enough to determine the alignment of T-counters and RF time, because T-counters can possibly be alignment to a wrong RF time bucket. Specifically, even if these two plots look perfect, some T-counters can be off by multiples of $2.004ns$. Alignment to the right RF time bucket is checked with

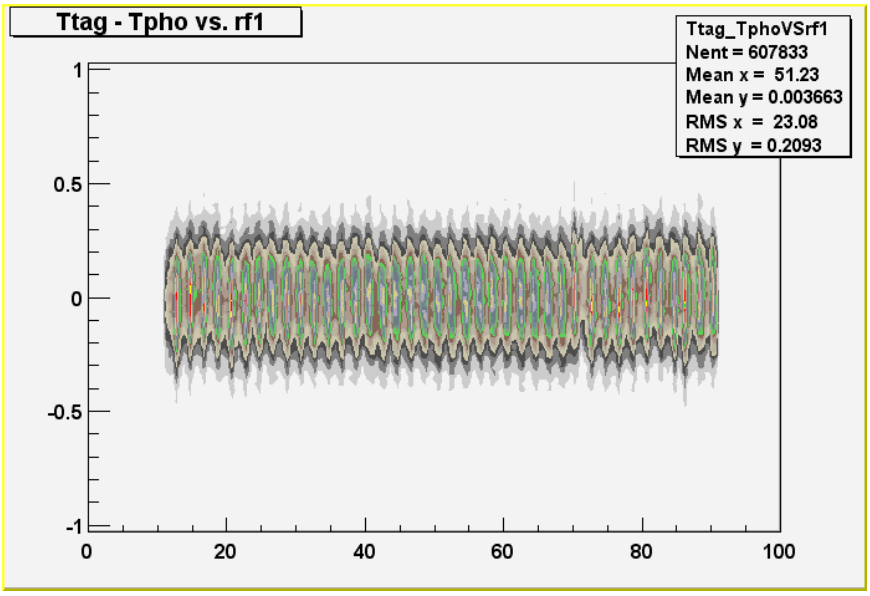

Figure 5: Calibrated RF time.

Histogram Name	Y-axis / X-axis
RF	Ttag - Tpho / T-bin ID
TagRF	Events / Ttag - Tpho
REF_ci	Tpho - Tst1 / T-bin ID

Table 2: Histograms for checking tagger/RF time alignment.

histogram REF_ci. This plot shows directly the delay of each T-counter relative to the reference detector[6]. Therefore it tells you if any of the T-counters is off by one or several time bucket, see picture 7.

The Tagger-CLAS offset is controlled by one constant(tag2tof) in the tagger reconstruction software. Previously, this constant is determined in the tagger calibration by comparing the tagger time and the start counter time. However, there is another offset between start counter and CLAS. Besides, there is no procedure to calibrate the start counter. So at this point, I am taking tag2tof out of tagger calibration and combining it with the TOF paddle2paddle delay calibration.

[6]Usually the start counter is used as the reference detector. In special cases, other detector can be used. See the `-R` option.

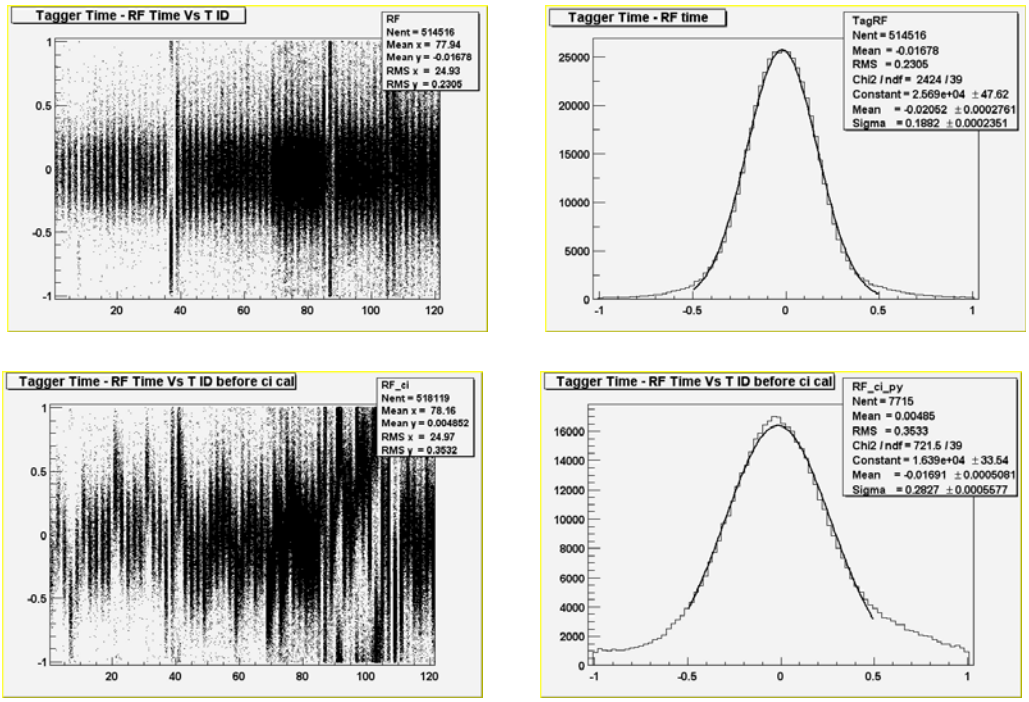

Figure 6: The alignment between the tagger time and the RF time.

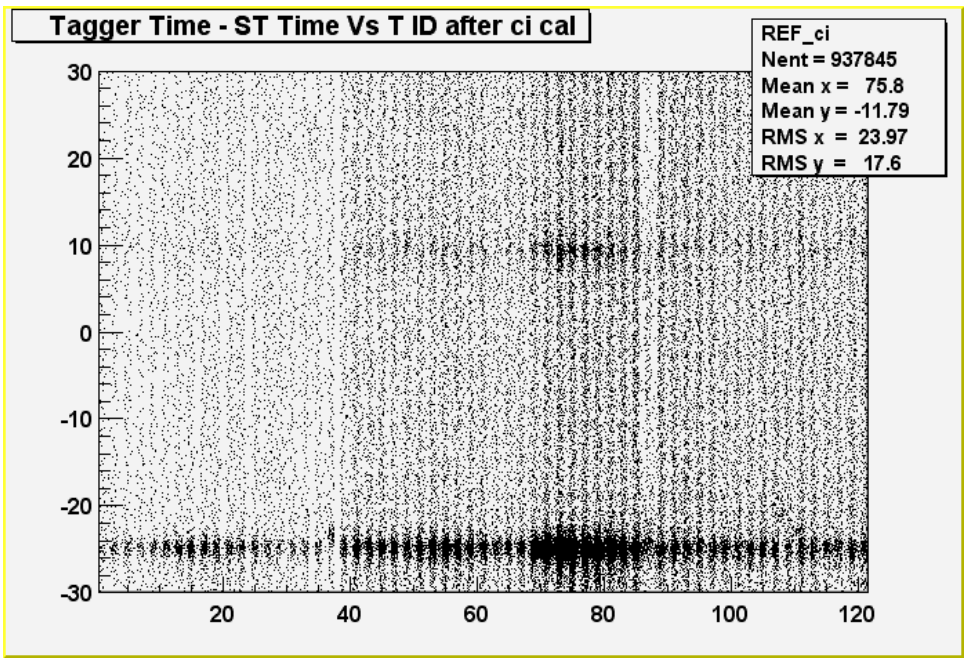

Figure 7: Relative delay between the tagger and the start counter. The two bands are due to the two triggers used in the experiment.