**2015 Shower Study Revision – Summary of files modified and documentation.**

This documentation includes reviewing all the steps before and including the shower analysis. The motivation was that the shower study did not find events when it should. Mark Adams ran calculations manually and had results that the shower study needed to match. We had golden files for all our tests.

This is the list of programs fixed/updated and a brief explanation of what was changed/added:

|  |
| --- |
| 1 - /cosmic/src/perl/**Split.pl**  Split.pl parses the rawdata uploaded by any user and splits it by date according to the Julian Day of the first event. Fixes: |
| * **Calculation of cpldfrequency**:   The cpldfrequency (25MHz for 6000 series and 41.7MHz for earlier series) is an important value for the creation of the threshold file, which is the building block of analyses.   * + Sample lines from rawdata:   CEB1409A 80 00 37 00 00 00 36 00 CE039834 204936.026 081014 A 03 0 +0053  D7C20D9C 80 00 3B 00 00 00 3B 00 D6F469B4 204943.002 081014 V 02 0 +0037   * + **Problem we found:** The cpldfrequency is calculated by examining the prior line ticks (CE039834), the new line ticks (D6F469B4), the prior time in seconds (204936.026) and the new time in seconds (204943.002).   There are two guesses, formulas:  Guess 1:  $cpld\_freq = $fg1 + (($dc - $fg1\*$dt + $Nover2) % $N - $Nover2)/$dt;  Guess 2:  $cpld\_freq = $fg2 + (($dc - $fg2\*$dt + $Nover2) % $N - $Nover2)/$dt;  These values are collected in arrays and then there is a statistical analysis to derive the cpldfrequency.  We found out that this works except when the clock is lying so we wrote code to detect if the clock is not right and exclude the cpldfrequency guess from the array because these values were sometimes off by a little and they would contribute to yield a cpldfrequency that was not right. See the following example where the clock reported that 7 seconds elapsed when in reality 6 seconds had elapsed (calculations from data lines above):  old dt1: 7 old calc1: 21428571.4285714 new calc1: 25000000 new dt1: 6  old dt2 7 old calc2: 21428571.4285714 new calc2: 25000000 new dt2: 6 |
| * Mihael’s explanation of the formulas above:   **Notation:**  N = maximum clock tick value (i.e. ticks are always mod N)  dC = difference in number of clock ticks; we are only able to measure (dC mod N)  dt = time difference, in seconds  fg = CPLD frequency guess  Now, given some frequency guess, after some dt, we expect to see  dC\_guess = dt \* fg mod N  If our guess for fg is correct, we also expect that  dC\_guess - dC\_actual = 0  In other words:  dt \* fg mod N - dC mod N = 0  or  (dt \* fg - dC) mod N = x\_i, with x\_i = 0.  This is, of course, somewhat approximate. It is entirely possible for actual values to be only very close to what they should be, but not exact, so we need to devise a statistical test.  Using the formula above is problematic since if dt \* fg - dC == -1, we get (dt \* fg - dC) mod N = N - 1. If we were to compute something like a standard deviation from the x\_i, a small difference will lead to a large contribution to the standard deviation. For example, we could have x\_i = {1, 0, N - 1, 0, 0, N - 1,...}.  That's dealt with by shifting things by N / 2, such that small differences under mod N translate into small differences outside of it:  x\_i' = (dt \* fg - dC + N / 2) mod N - N / 2  Then, for small variations like above, x\_i' = {1, 0, -1, 0, 0, -1, ...}.  If our fg was wrong, then we would get roughly random stuff, say  x\_i' = {10, 90993021, -1421344, 12039, 22333,...}  The - N / 2 after and +fg1 (or fg2) in front are not very relevant as far as the standard deviation goes. But for small deviations of the actual frequency from the nominal one, you would get x\_i' with a mean that isn't exactly zero. When you add the guess frequency, the shifted mean would generally give you a correction to the nominal frequency.  For example, assume that there was a small drift between the guess frequency and the actual frequency:  f\_actual = fg + epsilon  Then the actual counts would be:  dC = (fg + epsilon) \* dt mod N  Defining (this is what's in Split.pl):  y\_i = fg + [(dt \* fg - dC + N / 2) mod N - N / 2] / dt  we will have:  y\_i = fg + [(dt \* fg - (fg + epsilon) \* dt + N / 2) mod N - N / 2] / dt      = fg + [(dt \* fg - dt \* fg + epsilon \* dt + N / 2) mod N - N / 2] / dt      = fg + [(epsilon \* dt + N / 2) mod N - N / 2] / dt  Again, for small epsilon (specifically |epsilon \* dt| < N / 2), the mod N goes away, so:  y\_i = fg + [(epsilon \* dt + N / 2) - N / 2] / dt = fg + [epsilon \* dt] / dt       = fg + epsilon = f\_actual  So then the mean of y\_i would be a pretty good guess for f\_actual.  Anyway, that's the gist of it. |
| * **Calculation of time in seconds:**   To calculate the cpldfrequency, the code uses word 11 (204936.026) which is displayed as hour, minute, second and fraction of a second. This value is converted to seconds. For the series 6000 this is all that is necessary. For earliers series, we need to take into account word 16 (+0053) as an offset.   * + **Problem we found:** the offset was added without checking the series. |
| * Mark Adams’ explanation of how this all works:   The Perl code is doing the wrong thing.  We agree that the word 11s are different in the two lines  If the code is adding word 16 and then Checking, that is incorrect.  I had pointed out to Tom last summer or fall (and he corrected the code) that word 16 is not useful in 6000 DAQs.  Perhaps the same calculation was used in a different routine as well and it was only fixed in one place.  The important issue is that word 11 contains information to get the correct second.  The extra digits are used in some DAQ versions to correct the buffered value BUT only the integer value of the second, i.e. no fractional part of the second information, is used to calculate the absolute time.  We discussed this change of word 11 across DAQ versions earlier this week but I may have stressed the role of latching the GPS info too much to communicate the point.  I BELIEVE, but have not verified all situations that this is the way the second is determined via word 11.  Pre-6000 DAQ  GPS is latched into a buffer once per second.  Word 16 keeps track of how much time elapses until between GPS a signal arrival and it being written into the buffer of word 11.  I'm not sure of that statement but I think I'm know how word 11 and 16 are handled to get the correct time second. Word 16 is added to word 11 and rounded up.  That corrects all of the events where word 11 had not been updated with the new time from the GPS before the trigger occurred.  6000 DAQ  Word 11 is latched for every event.  The second is supposedly correct and word 16 becomes useless.  Sten will have to be consulted about details of how this fixed the latching delay issue.  I have forgotten.  We (Mario) discovered that there is still a delay problem and word 11 is not updated until 70ms after the 1pps arrival.  This characteristic bug is what we are trying to find in data after the fact to catch the pre-12 firmware versions.  6000 DAQS firmware version 12  To fix the 70ms hole, info from 1pps delayed almost a full so all values of word 11 have the previous second and 1 sec is added for all events in software in WireDelay.  Word 16 is still useless.  My understanding of the situation may still be flawed but it has gotten us this far.  As we say is science, it isn't the truth but it is the best description of reality we have ... Until the god of Sten weighs in.  Bottom line: 6000 DAQ time calculation code should not use word 16 for anything.  If we ignore word 16 your code would have found the correct firmware version.  It's not a problem with your algorithm, except that word 16 is being used. |
| * **Data lines that were not included in the split:**   When Split.pl reads the lines from the rawdata, the code makes decisions about what to do with each of them.   * + **Problem we found**: many lines were rejected and not included in the split file.   Reasons why they were rejected:   * + - A stuck gps latch     - A stuck clock     - The clock advanced but the gps latch did not     - The gps advanced and the clock did not     - Word 15 had a 08 flag   We removed those conditions, they lines now are in the split file and the code writes an entry into the error file that has the same name as the source file. |
| * **Splits were failing due to wrong channel count comparison for ST 3 cases:**   The code checks channel counts through analyzing the ST lines and the channel counts in the new line as compared to the previous ST line.   * + **Problem we found:** The ST 3 command resets the counts so there are situations when the new count is the same as the count before. The old code only accepted counts that were different. Consequence: the whole split file failed. |
| * **The whole upload would fail if there were “in-between” failed split files:**   The rawdata may contain several days worth of data that will be split through the code.   * + **Problem we found:** if one of the “in-between” days failed, split failed the whole upload and told the user that it had created a certain number of usable files but they were never accessible by the user because the failure prevented the code to add the metadata to the database. |
| * **Firmware calculation:**   The split code reads the firmware from the ST lines. The firmware is used to create the threshold file as well as in the wiredelay calculation.   * + **Problem we found**: We allow uploads without requesting that the rawdata has ST lines. Mark and I have been working to try to figure the firmware out from the actual data???? To be continued… |
| * **Split code was not cleaning up when splits failed:**   The split code will still fail some uploads that do not meet the required conditions.   * + **Problem we found**: The code was not cleaning up after a failed upload leaving orphan files in the server. |
| 2 - /cosmic/src/java/gov/fnal/elab/cosmic/analysis/**ThresholdTimes.java**  This code calculates the absolute time of both the rising edge and falling edge of an event down to ¾ of a nanosecond. |
| * **ThresholdTimes code needs to check for the firmware in the time calculation:**   In the formula to calculate edgedtime:  edgetime = rePPSTime[channel] + differenceOverCPLDFrequency + tmc / (CPLDFreq\*32)  if the DAQ is 6000 series and the firmware is less than 1.12 and the differenceOverCPLDFrequency is less than 0.07, then we need to add a second to that total before it is plugged into the calculation for edgetime. |
| 3 - /cosmic/src/perl/**WireDelay.pl**  This code calculates the data offset by cable length amount. |
| * **Wiredelay needs to add the firmware offset and the cable length if the DAQ > 5999 and the firmware > 1.11:**   The code had a check to add the firmware offset if the DAQ > 5999 and the firmware > 1.11 but this snippet of code forgot to add the cable length that had just been calculated. |
| 4 - /cosmic/src/perl/**EventSearch.pl**  This code finds showers based on the rising edges. |
| * **EventSearch.pl did not include the last line from input**   Perl behaves differently in loops and in this case, the code needed to process the last line of input when the loop that reads the file is done. It was missing the last event if there was one. |
| * **We added Multiplicity:**   This means which counters have been hit regardless of how many times they have been hit. |