Implementation of a Trigger Algorithm for SuperCDMS

Chiara Magliocca

September 30, 2017

Abstract

This report describes the operation of a trigger algorithm, implemented for SuperCDMS. An overview of the experiment and general informations about its trigger system are outlined in Section 1. The description of the code is provided in Section 2. The study of the algorithm efficiency is presented in Section 3.

1 The SuperCDMS SNOLAB Experiment

Substantial evidence supports the existence of vast amounts of dark matter at all scales in the Universe. Although its nature is still unknown, the possibility that dark matter is made from elementary particles beyond the Standard Model makes its detection and identication one of the greatest challenges of modern physics. Direct detection experiments attempt to spot the rare interaction of dark matter particles in our neighborhood of the galaxy with normal matter targets in deep underground laboratories. The expected signature is elastic scattering of dark matter particles with nuclei. These nuclear recoils must be distinguished from the abundant electron recoils due to interactions of normal matter particles in the targets.

Because a particle with electroweak-scale interactions and mass can be thermally produced in the early universe in the right amount to account for the observed dark matter relic abundance, Weakly Interacting Massive Particles (WIMPs) have been the main focus for the majority of direct dark matter searches.

1.1 The Project

The SuperCDMS SNOLAB experiment was selected by the US Department of Energy (DOE) and the National Science Foundation (NSF) to be a next generation (G2) dark matter search experiment, with the main science goal to improve the sensitivity for dark matter particles with masses 10 GeV/c^2 by at least one order of magnitude over existing sensitivities. The US agencies are also interested in having the experiment be capable of reaching sensitivities where solar neutrinos become a background for low mass dark matter particle detection.

The experiment will employ cryogenically-cooled germanium (Ge) and silicon (Si) crystals, capable of measuring the very small nuclear recoil energies produced by elastic scattering of dark matter particles with nuclei, and two different types of detectors:

1. To search for dark matter masses well below 1 GeV/c^2, with sensitivity primarily limited by electron recoil backgrounds
2. To measure both ionization and phonon signals from dark matter interactions and provide exceptional discrimination between a nuclear recoil signal and electron recoil backgrounds for dark matter particles with mass greater than about 3 GeV/c^2.

Six detectors, along with the associated mechanical, thermal and electronic support structure,are integrated into each modular Detector Tower.
For the SuperCDMS SNOLAB project baseline, we will have four towers (two with four Ge and two Si, one with 6 Ge, the other with 4 Ge and two Si). These four towers will be housed in a cryogenics system capable of operating up to 31 towers at temperatures as low as 15 mK.

1.2 Trigger System

The trigger will be a multi-level set of hardware and software algorithms that provides robust triggering capabilities for the experiment. The ultimate goal is to have stable triggering at as low of a threshold as is possible, an acceptable trigger rate and data volume, a fully efficient trigger once analysis thresholds are applied, the ability to acquire the necessary calibration samples, and minimal sensitivity to environmental noise. The trigger is also intended to operate without deadtime and it must support running in a number of modes including:

- low-background running (WIMP search data)
- high-rate calibration runs
- acquisition of noise traces via random triggers.

The trigger needs to take in data from a number of sources including the two different detectors and, at the end, we must be able to read out the experiment in a number of congurations that include:

- Reading out all the towers in the experiment for the same trigger time
- Reading out just a single detector or tower that passes the trigger

2 The Code

The aim of this algorithm is to select the data that we can consider valid for a further offline analysis, starting from a sample of data in ADC values taken directly by the detector without any hardware trigger.
The algorithm discriminate good pulses from noise thanks to a threshold, set after an accurate study of noise distribution, and examine the eventuality of contamination between pulses that occur in the same event or in two different ones.
As a result we have two two-dimensional lists: the first one includes the current values of the events that passed all the filters and are considered "good" so far, while the second one includes the indexes of these events.

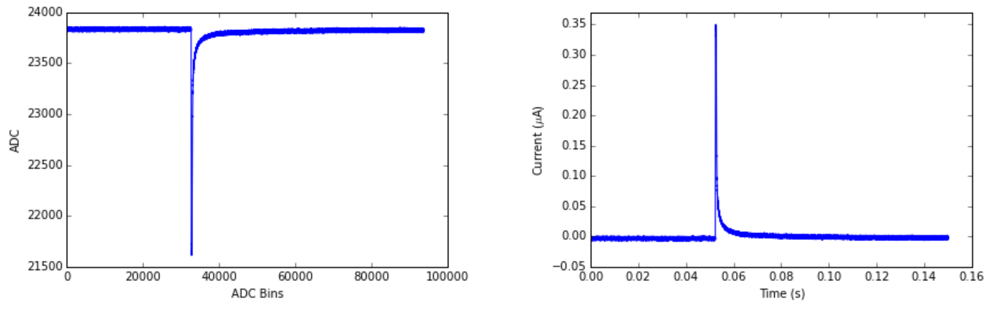
An event can be flagged as "good" if:

- It contains at least one pulse
- All the pulses start and end within the acquisition frame
- No pulse is contaminated by another one that occur in the same event
- No pulse is contaminated by another one that occur in the previous event

2.1 Study of the noise distribution

After reading the ADC values data from a file, the algorithm stores them in a two-dimensional list called res(). Data are then converted from ADC values to current, centered to zero and reversed (so that the pulses will be positive and the subsequent studies simplified) and stored in a list called pulse().
Figure 1a shows an example of event stored in res(), while figure 1b is the plot of the same event, with values converted to current.

As we just said, an event can be considered valid for a further analysis if at least a pulse is present. But how can we know if that pulse is actually a signal or just noise? We need to set a threshold and discriminate pulses from noise. To do this a study of the noise distribution in needed. The threshold will be set at 3σ.
We start selecting all the events that contain no pulses, but just noise. To do this, a temporary threshold is set to 0.02 μA (this value has been chosen after an analysis of a sample of events with both noise and signal, realizing that the major part of the pulse exceed this value). The indexes of the events that contain just noise are then stored in noise(), while the events that contain both



(a) (b)

Figure 1: (a) Plot of res [315]. Plot of the event 315 as it is taken by the detector without any hardware trigger.
(b) Plot of pulse [315]. Plot in current(μA) of the same event, centered and reversed.

noise and signal are stored in nonoise().

At this time the standard deviation (σ) of each just-noise event is calculated to determine, at the end, the mean value of all of these. The threshold is then:

$$A_{th} = 3\sigma \approx 0.005 \tag{1}$$

Figure 2 shows the plot of a just-noise event. The red line is the trigger threshold $A_{th} = 3\sigma$.

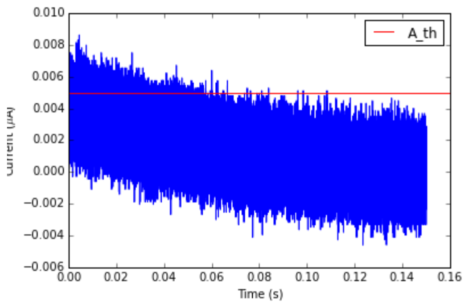
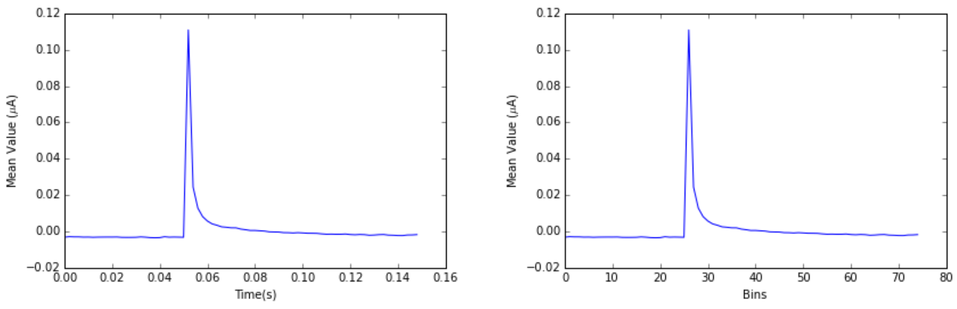


Figure 2: Plot of a just-noise event. The red line is the trigger threshold equal to 3σ

2.2 Low-Pass Filtering and Moving Average

What we need to do now is low-pass filter the pulse to get rid of high-frequency information which are not actually useful for this algorithm. To do this, we calculate the moving average.
Each event is divided into intervals of fixed length (=sample period) T_{sample} (at this time chosen to be 0.002 s) and the mean value of the signal in each interval is calculated (to clean it up from additional noise and smooth it). T_{sample} parameter has been chosen equal to 0.002 since it seemed that the high-frequency information were reasonably cutted out. It can be change to optimize the algorithm, but, to understand its optimal value, an analysis of the algorithm's efficiency is needed.
The mean values of the intervals are stored in a two-dimensional list mean[i][j], where i is the index of the event and j the one of the interval.
Figure 3a shows the plot of the same event of figure 1b after the low-pass filtering process. Both the figures are plotter VS Time(s) for a better comparison.
Figure 3b shows the plot of the mean value of the event as a function of the number of intervals in which this is divided.

From now on, we will always use the filtered pulses.



(a) (b)

Figure 3: (a) Plot of mean[315]. Plot of the same event of figure 1b after the low-pass filtering and moving average process. Here the size of the pulse is lower than the one of the pulse on figure 1b because of the low-pass filtering process. For this reason, to study the characteristics of a pulse we will go back to the original plot and to the original values, to not lose part of the signal.
(b) Plot of mean[315] (μA). The x axes are the intervals in which the event is divided.

Is now time to check the number of pulses present in each event. To do so, the algorithm scans all the events and, in each event, checks whether a value exceeds the trigger threshold (the one that was previously fixed). If it does, it saves the index of the interval that include that value in a two-dimensional list called indexdata() and the value itself in findata().
In figure 4 is shown the plot of a filtered event. The green line represent the trigger threshold, while the red one are the values above threshold stored in findata().

At this time the algorithm looks for groups of consecutive indexes in indexdata() and saves in first() and last() respectively, the index of the values of the first and last signal above threshold (basically the first and last value to cross the threshold).
The length of first() (or equally the length of last()) is actually the number of pulses present in the event under examination.

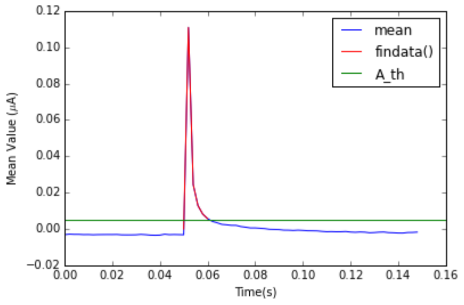


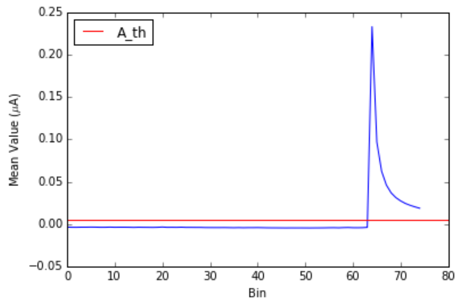
Figure 4: Plot of mean[315] (μA) VS Time (s). The green line is the trigger threshold while the red one are the values above trigger threshold stored in findata().

2.3 Pulse Contamination Study

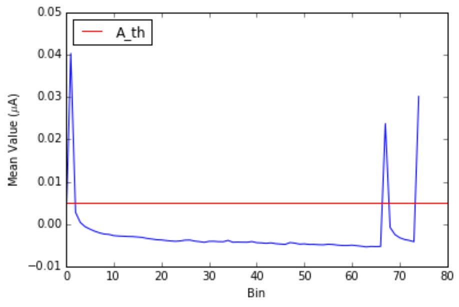
As we said before, we consider "good" an event in which all the pulses are not contaminated by another one that occur in that same event or in the one before. For this reason, to discriminate "good" event from "bad" event it is necessary to study the eventual contamination between pulses.

Uncompleted Pulses Before starting, the algorithm throws all the event with at least an incomplete pulse. Such event can occur either when the acquisition stops before the last pulse is finished (as is the case shown in figure 5a) or the pulse starts before the acquisition, so that we lose the beginning of the signal (as in the case shown in figure 5b).

To know if the acquisition stops before the end of the pulse the algorithm checks if the last value of the last pulse is still above the trigger threshold. In this case the pulse is not finished and the entire event is thrown.
If the first value of the first pulse is already above the trigger threshold the pulse started before the acquisition. Also in this case the entire event is thrown.



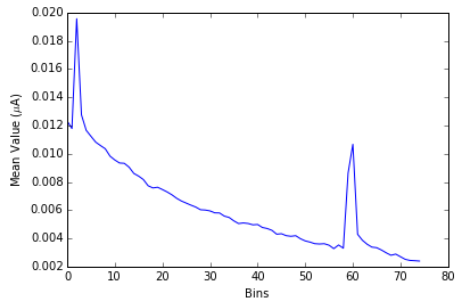
(a)



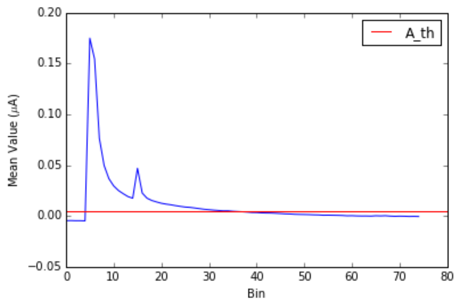
(b)

Figure 5: (a) Example of an event with an uncompleted pulse. The red line is the trigger threshold. The acquisition stops before the pulse is finished so that we lose the last part of the plot. The entire event is thrown.
(b) Example of an event with two uncompleted pulses. The first pulse starts before the acquisition so that we lose the first part of the signal, while the last pulse is not finished. Also in this case the entire event is thrown.

Contamination between pulses in two different events To understand if a pulse is contaminated by another one that occur in the previous event, the algorithm calculate the slope of the k points before 'first' (=first value above trigger threshold of that specific pulse). At this time $k = 6$ but, as the T_{sample} parameter, it can be optimized after a study of the efficiency of the algorithm.
If the calculate slope is negative it means that the pulse under consideration occur on the tail of a previous pulse (as shown in figure 6a), so the algorithm throws the entire event and start analyzing the next one, while if the slope is non negative it saves the index of the event in a list called indexgoodevent().



(a)



(b)

Figure 6: (a) Example of an event contaminated by a pulse that occurs in the previous event. In this case the slope is negative and the entire event is thrown. (b) Plot of an event in which the first pulse is contaminated by the second one. The second pulse occurs on the tail of the first one. The red line is the trigger threshold, thanks to which we check if the pulse is finished or not. The second pulse clearly occurs when the first one is still above the threshold.

Contamination between pulses within the same event At this time the algorithm studies the contamination between pulses that occur in the same event, scanning only the ones included in indexgoodevent(), to analyze just the pulses considered good so far.
A pulse is contaminated if another one occurs before it is finished (this means that another pulse is present on its tail). For this reason the algorithm check only the tail of the pulse, starting from the index of the interval that contain the maximum value of the current.
The check is based on the idea that, after the maximum value, the signal decreases, so each value should be greater than the subsequent one. If this condition is respected until the value of last() of that specific pulse for all the pulses in the event, the algorithm saves its index in indexresult() and the corresponding current values in result(), otherwise the entire event is thrown. Figure 6b shows an event in which the first pulse is contaminated by the second one, that occurs on its tail. In figure 7 is plotted an event that is considered good so far, as every pulse starts and ends within the acquisition frame and no pulse in contaminated by any of the two ways.

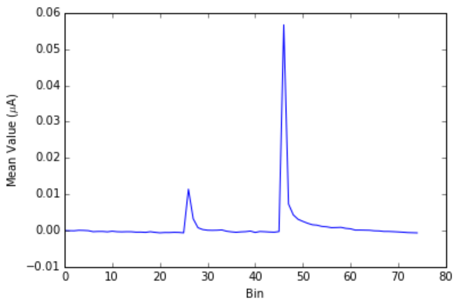


Figure 7: Plot of mean[201]. Example of an event that can be considered good so far. Both the pulses in this event start and end within the acquisition frame, and no one of them is contaminated.

At the end, indexresult() contain the indexes of the clean events (the events in which at least a pulse is present, all the pulses start and end within the acquisition frame and with no contamination), while in result() are stored the current values of each of these event. We can now study the efficiency of this algorithm to also select the optimal k e T_{sample} parameters.

3 Study of the Efficiency

I decided to study the efficiency of this algorithm with both real and fake data.
For real data, I looked at the plots of a sample of events as they are taken by the detector (res()), and I selected by hand the pulses that, to me, were good and bad. I stored the indexes of the events in two different lists and I used the algorithm on my selected events.
To study the efficiency with fake data, I implemented an algorithm that produce data following our pulse distribution and I used the algorithm on different sample size to obtain, at the end, the efficiency curve.

3.1 Real Data

Tables 1a and 1b summarize the real data efficiency for two different sample size: in table 1a sample size is 100 for both bad and good events, while in table 1b the sample size for the bad events is 280 (sample size for good event is still 100). Results are plotted in figure 8.

Sample = 100	Good	Bad
Good	50%	50%
Bad	17%	83%

(a) Table of the efficiency for real data with sample size good and bad = 100.

Sample Bad = 280	Good	Bad
Good	50%	50%
Bad	20.7%	79.3%

(b) Table of the efficiency of the algorithm when using real data with sample size good = 100 and sample size bad = 280.

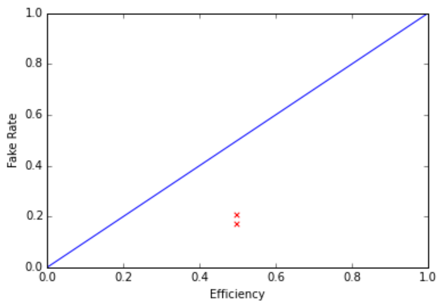


Figure 8: Plot of the efficiency of the algorithm using real data.

3.2 Fake Data

Table 1 summarizes the efficiency of the algorithm when using fake data. As we can see, the reasonable sample size to study the efficiency is 1000, as the fluctuations of the results decrease. The results are shown in figure 9.

Sample = 100	Good	Bad
Good	98%	2%
Bad	40%	60%

(a)

Sample = 500	Good	Bad
Good	98.8%	1.2%
Bad	72%	28%

(b)

Sample = 1000	Good	Bad
Good	99.2%	0.8%
Bad	71.7%	28.3%

(c)

Table 1: Table of the efficiency for fake data with sample size = 100, 500 and 1000.

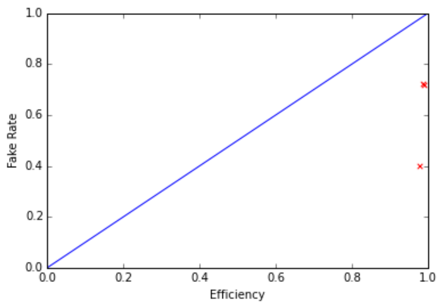


Figure 9: Plot of the efficiency of the algorithm using fake data.

4 Conclusion

As we can see from the results of the study of the efficiency, the sample size used to study the real data was not enough and, approximately, 1000 events are needed to have a reasonable result.
In the study of the efficiency using fake data, the number of bad event flagged as good is pretty high. This because the algorithm that produced the data is simple, the data produced are approximate and do not follow the true data trend perfectly.

As a result, both the algorithms can be largely improved, both to better produce fake data and properly study the efficiency, and to better discriminate valid events from spurious ones.
To have a better efficiency we can:

1. Change free parameters: the temporary threshold that we use to study the noise distribution, as well as the definition of the trigger threshold itself and the two free parameters T_{sample} (sample period in which we calculate the moving average) and k (number of intervals in which we calculate the slope) can be changed and optimized.

2. Re-define last(): the list last() is filled with the last value of each pulse that cross the trigger threshold, but we cannot really know if at that time the pulse is finished or not. If it is not, another pulse can occur at the very end of its tail (after the value stored in last()), as in the case shown in figure 10, and the algorithm will not be able to flag that pulse as contaminated. To overcome this, we might define the members of last() starting from the ones of first() (list in which it is stored the index of the interval that contain the first value above the trigger threshold of each pulse of the event), taking into consideration the time-lenght of a generic pulse.

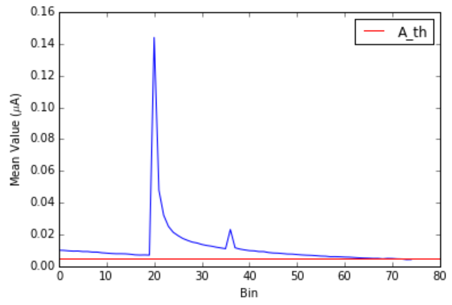


Figure 10: Example of event in with the second pulse occurs at the very end of the tail of the first one. The algorithm is not able to flag this event as "contaminated" and the event is stored in indexgoodevent().

3. Target better the uncompleted pulses: the algorithm compares to the edges of the acquisition frame only the really first or last value of the pulse, but, for some reasons, it can happen that, even if the pulse in not finished, the last value of the event is below the trigger threshold and the event is then flagged as "good". So, as the algorithm is today, it is not able to detect all the uncompleted pulses and sometimes bad data are flagged as good data. This study should be implemented to be sure all the pulses that we have start and finish within the acquisition frame.

4. Split good traces: as the algorithm is today, if a pulse is contaminated or if it is not finished, we throw the entire event. But this is not what we want at the end. If, for example, three pulses are present in a event and two of them are contaminated, we can still save the third one, and study that one. We have a lot of pileup, what we want is to either reject a single pileup or split good traces into independent events if separated by some minimum distance. Figure 11 shows an event in which the first and second pulses can be considered good, while the third one should be flagged as bad as it is uncompleted.

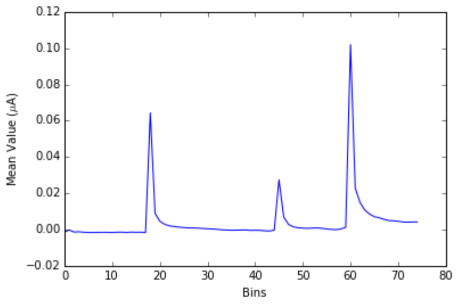


Figure 11: Example of an event in which the first and second pulses can be flagged as good, while the third one should be thrown as it is uncompleted.