Tuning of the MuPix8 High-Voltage Active Pixel Sensor

Diego Mauricio Salgado Llamas

Bachelor thesis in Physics
Presented to the Fachbereich 08 – Physik, Mathematik und Informatik
at the Johannes Gutenberg-Universität Mainz

November 29, 2018

First evaluator: Prof. Dr. Niklaus Berger
Second evaluator: Prof. Dr. Lucia Masetti

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Mainz, den 29. November, 2018 _____

Diego Mauricio Salgado Llamas
AG Berger
Institut für Kernphysik
Staudingerweg 9
Johannes Gutenberg-Universität
55128 Mainz

Abstract

The MuPix is a high voltage monolithic active pixel sensor (HV-MAPS) designed for the Mu3e experiment. This experiment conducted at the Paul Scherer Institute is searching for lepton flavour violating decay of the $\mu^+ \longrightarrow e^+e^-e^+$, which is not allowed in the standard model. The goal of this thesis was to minimize the position dependent signal delay, that in turn affects the time resolution of the detector. As a part of the thesis, an algorithm was developed to perform automatic threshold scans. To measure the time delay of a signal, a method was implemented by injecting signals to the pixels. This injections allow to do test measurements without the need of signals from actual particles. The range of the threshold tuning (individual adjustment of the global threshold) was estimated to be $(41.521 \pm 0.150)\mathrm{mV}$. Several attempts to alter the delay of the injection signal with different configuration of the injections and the tune values for the high threshold were not achieved within the current available range of the MuPix8. This result indicates that the pixel to pixel delays in the signal are not caused by time walk and are not amplitude dependent.

Zusammenfassung

Im Rahmen vom Mu3e Experiment am Paul Scherer Institut wurde der MuPix Sensor entwickelt. Dieses Experiment sucht nach dem $\mu^+ \longrightarrow e^+e^-e^+$ Zerfall. Dieser Zerfall ist laut dem Standardmodell, wegen der Verletzung der Leptonfamilienzahl, nicht erlaubt. Ziel dieser Arbeit ist es, die ortsabhängige Signalverzögerung zu minimieren. Diese Verzögerung beeinflußt die Zeitauflösung des Sensors. Als Teil dieser Arbeit wurde ein Algorithmus entwickelt, welcher das selbständige Scannen der Schwelle ermöglicht. Um die Zeitverzögerung des Signales zu bestimmen, wurde eine Methode implementiert, welche auf Einspeißung von Testsignalen basiert. Diese Testsignale erlauben eine Messung ohne einen Teilchenstrahl. Der Verstellbereich der individuellen Schwellenwerte wurde auf $(41.521 \pm 0.150)\mathrm{mV}$ geschätzt. Trotz mehrerer Versuche, konnte die Verzögerung des Signals nicht variiert werden. Die Ergebnisse zeigen, dass die Signalverzögerung zwischen einzelnen Pixeln nicht von der steigenden Flanke des Signals abhängt.

Contents

1. Introduction

The advancements in modern particle physics can in part be attributed to the advancements in the modern technologies that allow for the detection, identification and analysis of radiation. But often in science, the requirements for an experiment exceed the capabilities of current technologies. In these cases, it is up to the investigators, to develop new technologies that fulfil those requirements. This is also the case for the topic and subject of investigation in this thesis: the MuPix8.

As part of the Mu3e experiment at the Paul Scherer Institute the MuPix, a silicon based high voltage monolithic active pixel sensor, was designed. This particle detectors offer excellent vertex and time resolution. The MuPix8 is the first large prototype of the MuPix series. And like every new prototype with so many new features, there are bound to be some issues.

One of the issues is position dependent signal delays. This means that even simultaneous events could be given different timestamps and thus reduce the time resolution of the sensor. The topic of this thesis is to investigate how the size of this sensor affects its time resolution and try to improve the resolution by applying threshold tuning.

2. Particle detectors

Across many fields of physics, particle detectors play a central role. They allow for the study of different structures, levels of radiation, particle composition and many other things. There are different particle detectors with specific applications.

2.1. Semiconductor detectors

One of the types of particle detectors is the semiconductor detector. As the name indicates, the material used for these detectors is a semiconductor crystal. They rely on the small energy gap, the energy difference between the valence band and the conduction band, to operate. The energy gap in a typical semiconductor is about 1 eV. This semiconductor can be cooled down (e.g. to the temperature of liquid nitrogen) making sure that all the electrons are not thermally excited and remain in the valence band. The incident radiation could transfer enough energy to one of this electrons, to make the jump to the conduction band. With an electric field and the proper electronics, this charge can be collected and the radiation is detected.

Instead of cooling a regular semiconductor, one could also use a reverse biased semiconductor diode to avoid thermal noise. The diode is a semiconductor crystal with a so-called p–n junction. To create a p–n junction, one has to first has to have to separated parts of the crystal with different doping. A semiconductor, like silicon, is doped by adding impurities to the crystal (normally one impurity for every million lattice atoms [1]). The n–doped or n–type semiconductor has impurities with an extra electron, which is more weakly bound. The counter part of this is the p–type semiconductor, which has impurities with an electron less. The absence of the electron, called hole, allows the charge movement since electrons from adjacent atoms can move to the hole easily.

In a zero biased p-n junction, these two types of semiconductors are brought together. The diffusion current forms as the electrons from the n–semiconductor and the holes of p–semiconductor recombine. After the recombination, the impurities are left electrically charged. This means, the p–side has a net negative charge and the n–side has net positive charge, since the hole is now occupied by an electron, the p–impurities now have an extra electron, and vice versa for the n–impurities. The electrical field produced by this charges opposes the diffusion current and an equilibrium is reached. The electrically charged area at the junction is called the depleted region.

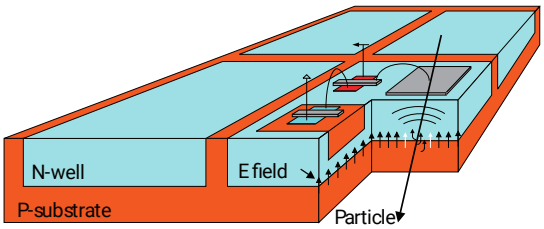

Figure 2.1.: Schmatics of a HV-MAPS [3]

If one applies a voltage to a diode so the n–part is connected to the positive terminal of an electrical supply, one says that the diode is reverse biased. Depending on the strength of the applied voltage, the size of the depleted zone can be modified.
These detectors have higher density than other detectors (e.g. gas detectors). This leads to a higher energy loss per ionizing particle (which allows for thinner sensors) and the creation of more electron–hole pairs (compared to the ion–electron pairs in a gas detector). The result is a larger number of charge carriers for the same amount of energy, which leads to a better energy resolution. Another advantage is the high mobility of the charges in the semiconductor, which allows the charges to be collected very quickly, making them ideal for experiments with high event rates.

One member of this group of detectors is the silicon detector. Since many electronic components are built using silicon, the implementation of silicon to built a detector allows for a good integration with the surrounding electronics [2].

2.2. HV-MAPS

In 2007 Ivan Perić proposed a new type of particle detector. The detector is a high-voltage monolithic active pixel sensor (HV-MAPS). The sensor itself is divided into multiple pixels. Each pixel of the sensor, as shown in Fig. 2.1, implements a deep weakly n-doped well in a p-doped substrate. When the junction is reversely biased, the depleted region becomes relatively thick, even to the point of overlapping with the neighbouring pixels and leaving no insensitive areas. The passing through of an electrically charged particle, generates electron — hole pairs. Thanks to the high voltage applied between the substrate and the n-well the collection of the charges is done by drift and not by the much slower diffusion. The read-out is done by the CMOS (complementary metal-oxide-semiconductor) logic inside the n-wells. In particular, every pixel has an amplifier inside the wells [3].

As part of the Mu3e experiment, an experiment looking for the lepton-flavour violating decay $\mu^+ \longrightarrow e^+e^-e^+$, the MuPix was proposed. The MuPix is a silicon based pixel detector based on the HV-MAPS technology, optimised for untriggered running.

2.3. MuPix8

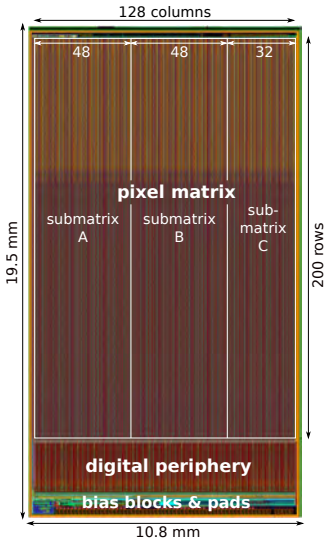

Figure 2.2.: The MuPix8 layout.

There are several versions of the MuPix to date. The one used during this project is the MuPix8. These sensors can be made really thin (to values below 50 µm) and have a fast charge collection and built-in zero-suppression [4].

The MuPix8 is composed of three matrices. The first matrix, matrix A, has 48 columns, the same as matrix B. The matrix C has a total of 32 columns. During this project, only the matrix A was investigated. The size of each pixel is 80 µm × 81 µm. This makes the MuPix8 the first large prototype of the MuPix series with an area of 1 cm × 2 cm [5] (see Fig. 2.2). It is still being investigated which effects the large size of this prototype has on its performance.

2.3.1. Readout

As shown in Fig. 2.3, each pixel has an amplifier. The amplified signal is driven in two different ways from the pixel to the periphery. For the matrix A, the signal is voltage driven, while for matrices B and C the signal is current driven. For each column there is a readout block with 200 cells, one for each pixel in the column. These readout blocks are on a non-sensitive area of the sensor (digital periphery in Fig. 2.2). A given cell receives the output signal of its corresponding pixel and from it, it generates information about the hit: timestamp, row (pixel address is given in a tuple of column and row) and amplitude.

The hit information in the read out cells is retrieved by the end of the column block (EOC). The EOC just collects the first read out cell with a stored hit. In a similar manner, the information stored in the EOC is collected by the state machine. This is achieved by taking the stored information of the first EOC with a hit. Finally, the hit information is completed by storing the column number of the corresponding EOC [7].

In each column block only one hit can be copied to the EOC, which means if there are several hits in one column, only one is read out. This happens simultaneously for the different columns, so afterwards, in the case of multiple hits, there should be several EOC storing hit information, which are read out one by one. When this is done, the

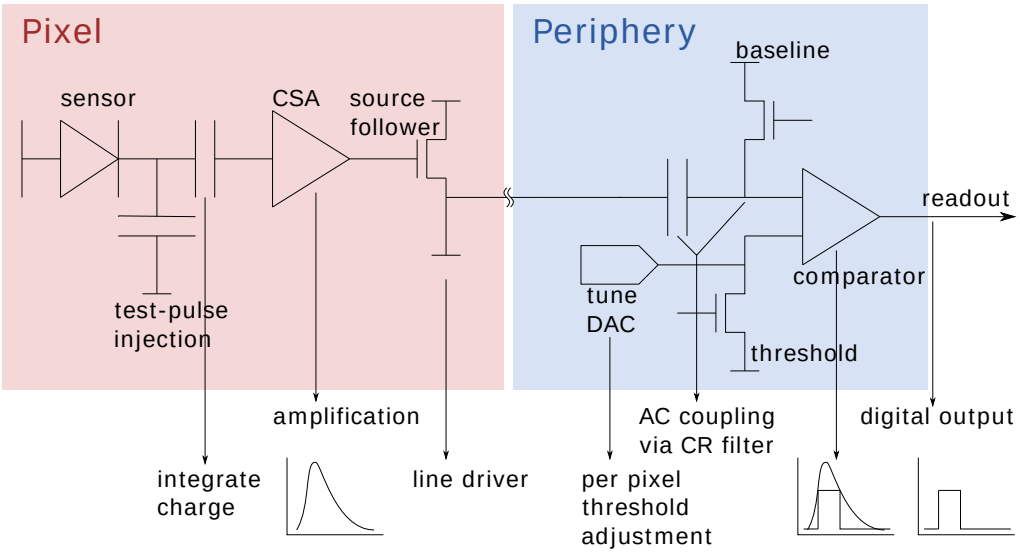

Figure 2.3.: Schematics of the electronics of the pixel and the periphery [6]. This diagram was originally designed for the MuPix7 (here modified). It shows the main components of the pixel electronics, with the exception of the two different form of tune DAC, TDAC1 and TDAC2.

next series of hits (one per column) can be read out following the same scheme.

The hits are then sent out serialised on a 1.25 Gbit/s low-voltage differential signalling (LVDS) link.

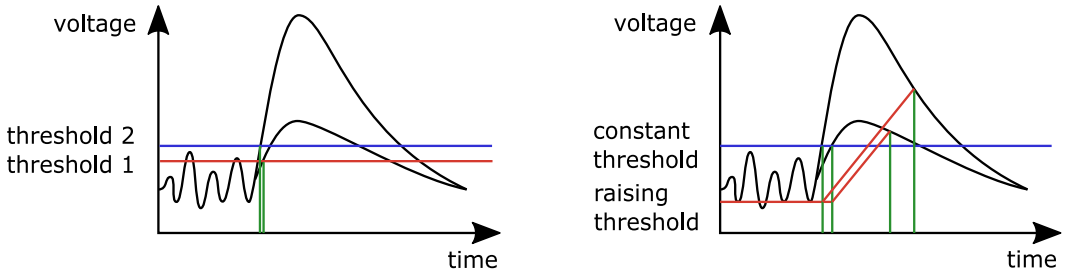

Figure 2.4.: Left: readout using the two threshold mode. Right: readout using the ADC threshold mode [5].

2.3.2. Time walk corrections

In the MuPix, to time a signal, a fixed threshold is used. When the pulse crosses over the threshold, the time is recorded. This timing technique is dependent on the amplitude of the pulse, as shown in Fig 2.5. For two pulses with amplitudes U_1 and U_2

respectively and $U_1 > U_2$, a fixed threshold should give time difference of $\Delta t = t_2 - t_1$, which should be positive, i.e. the second pulse is detected later because of its smaller amplitude. This Δt is known as time walk [9].

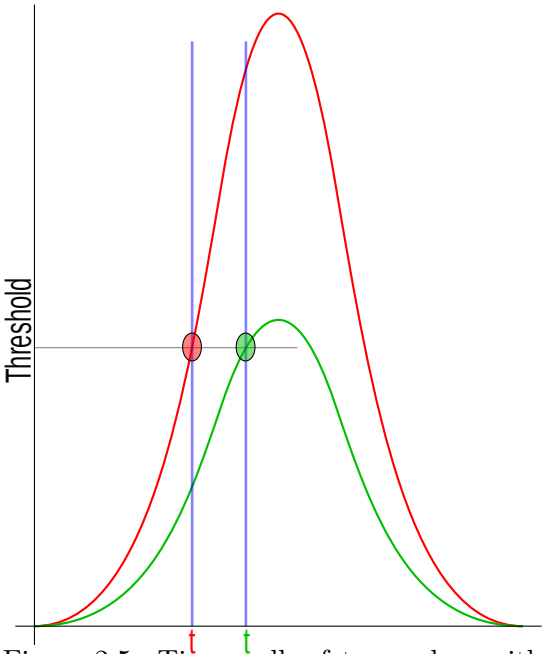

Figure 2.5.: Time walk of two pulses with different amplitudes while threshold triggering [8].

To correct this effect, the MuPix8 implements two different modes (see Fig. 2.4). The first one is the *two threshold* mode. As the name says, two thresholds are used, denominated low threshold (TH low) and high threshold (TH high). When the low threshold is crossed, the time measurement is taken. This can be set to a relatively low value (near noise level) to reduce the time walk. The second threshold is set to a higher value and is used simply to confirm that the signal that crossed the low threshold was in fact the searched signal. The second mode is the *ADC threshold voltage* mode. For this mode a constant threshold is used to time the signal. When this threshold is crossed, a timestamp is generated and a second linear rising threshold is activated. When the signal meets the ramp signal, a second timestamp is stored. This allows for a correction of the time walk by using the signal size.

2.3.3. Injections

For lab measurements, it is possible to simulate events by injecting a signal into any given pixel. This injection can be set for a pixel of choice with a given pulse width and amplitude. The pulses can be sent in a fixed number or constantly with a given frequency. The injections can also be sent to several pixels at the same time, covering areas of different sizes. It will be discussed, how efficiently large areas can be injected. But in principle, it is possible to configure injections for the whole matrix.

On the left of Fig. 2.6 a digital-to-analogue converter (DAC) is shown. This DAC decides the voltage V that is used to charge up a capacitor. When the switch is closed the pulse is then released, routed to a configured destination (a pixel or an area of pixels) and capacitively coupled into the pixel sensor. The L is a placeholder for a multiplexing logic. This logic decides which pixels are going to be injected.

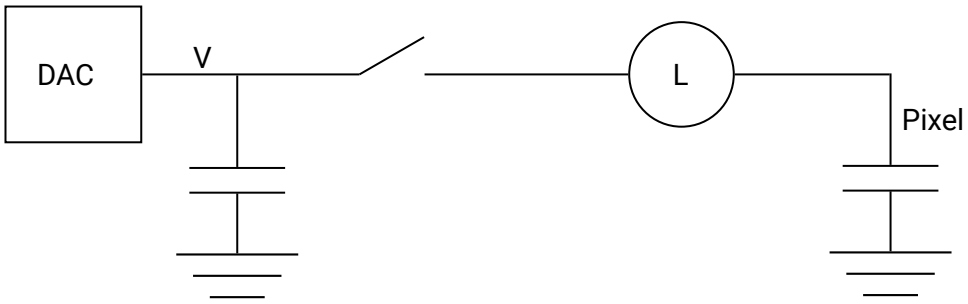

Figure 2.6.: Sketch of the electronics of an injection. The L stand for a complicated circuitry that reroutes the injection to a specific pixel address or a specific area of pixels.

2.3.4. Tuning

Because of its large size, the MuPix8 has lost some time resolution compared to its predecessor. Two simultaneous events appeared to register two different timestamps. One possible reason for this is the drop of voltage across the large area of the chip. This drop of voltage causes the amplifier in each pixel to produce pulses of varying sizes. Another possible explanation is that pixels that are farther away from the digital part have a longer electrical route. A longer line means higher capacities, which in turn means a longer charging curve. Combining these effects with timewalk could explain the delay in timestamps.

Tuning is the change of the threshold for every pixel individually. By applying changes in the individual thresholds the timing of the individual signals can be shifted to make the time differences between all pixels minimal and thus improve the time resolution of the sensor. As already discussed in section 2.3.2, each block cell has two thresholds. Tuning allows to manipulate these two values separately.

2.4. File structure

For this project, besides understanding the hardware, it is also important to understand how to use the software and the way that measurements are stored in the files that are analysed later. The measurements are stored in so called *block files*. The block files are an ensemble of telescope frames. The telescope frame is the container in which the hits are stored. And finally, each hit contains all the hit information,

i.e. pixel address, timestamp, etc. Besides of storing the hits, the telescope frame also has information about the triggers.

A typical measurement would be stored in a single block file. So if, for example, one wanted to create a hitmap (a 2D histogram that shows how many hits per pixel were registered) the block file containing the measurement would be accessed. The next step is looping over all the telescope frames, and from each frame collect all hits. Then, from each hit, extract the column and row information and finally add a count to the corresponding position on the hitmap.

As all the hits, the telescope frames also have their own timestamp. The clock of the blockfile has a frequency of 500 MHz, while the clock of the telescope frame has a frequency of 125 MHz. The hit timestamp is stored in the telescope frame with a 10 bit unsigned integer, meaning it has a range between 0 and 1023.

3. Experiment

3.1. The experimental set-up

In the scope of this thesis, the MuPix8 was investigated using a set-up containing the following parts:

1. MuPix8: the sensor and its properties have been discussed in section 2.3.

2. Printed circuit boards (PCB):
   The PCB handles the connections of the sensor to the periphery. This includes connections to power supplies with different voltage levels and signal connections to the FPGA. The sensor itself is bounded to small PCB (called insert). The insert can be easily plugged into the bigger PCB. See Fig 3.1.

3. Field programmable gate array board (FPGA):
   The FPGA is a piece of electronic capable of driving and receiving data with high rates. The FPGA plays an integral role in the read out system of the whole set-up.
   The FPGA has phase-locked loop from which it generates two synchronised clocks, one with 125 MHz and one with 500 MHz, which act as a reference for the sensor. It also generates the injection signals and sends them to the PCB. The FPGA sends the data over to the PC via PCI express. See Fig. 3.3.

4. PC: The PC receives the data from the FPGA, process it and stores in the hard drive. Therefore, a costume software package is available which includes a driver for the FPGA board, read out and control, as well as analysis software for the MuPix8.

5. Power: Three different voltages need to be supplied to the PCB board to the sensor. The high voltage source takes care of the fast charge collection.

3.1.1. Software

Throughout this project, all measurements were done using a piece of software called *single*. With *single*, one can configure injections, set board and chip DACs, mask or tune pixels and start and stop measurements. Most of the functions implemented during this thesis, rely on the original code of this software. *single* is part of the mupix8-daq repository.

Figure 3.1.: The MuPix8 on the PCB.

Figure 3.2.: Power supply and high voltage source

Figure 3.3.: FPGA and GPU

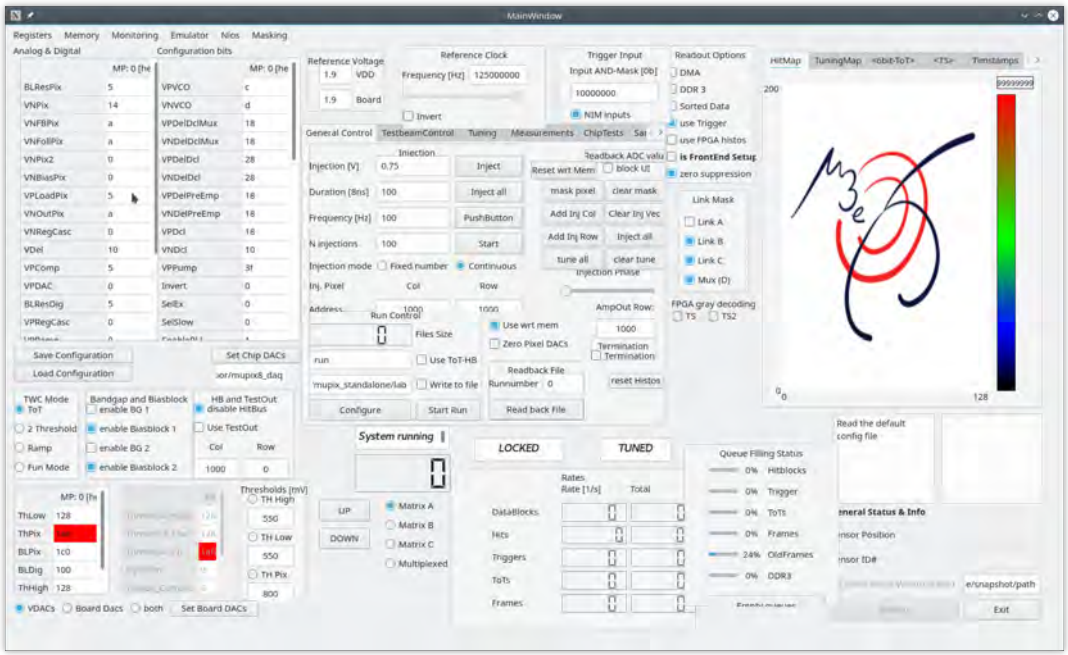

Figure 3.4.: Screenshot of the main window of the *single* software.

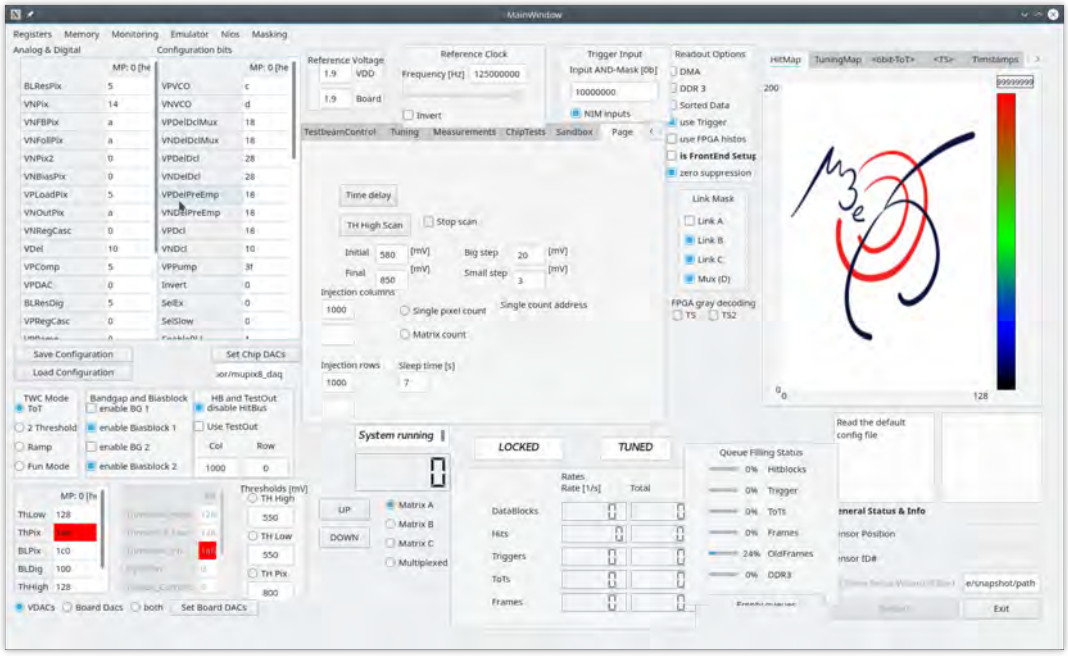

Figure 3.5.: Screenshot of the tab containing the functions implemented in this thesis.

3.2. First measurements

The first step of this experiment was acquiring some sense of how the position of the pixel affected its timing. Getting an initial notion of the problem was vital since it allowed the whole process that came to be more objective-oriented. The general idea was to measure how much the timing of an event changed across the area of the chip. The easiest way to this would be to simulate an event, using injections, and seeing how the signals changed, specifically how their forms changed, for different pixel locations. With the help of an oscilloscope, it was possible to measure the signal that was being injected into the pixel and compare it with the signal coming from the *hitbus*, the output of the comparator. Using the *single* software, the injection could be set to certain parameters, the most important right now being the pixel address. Additionally, with the software the hitbus had to be set to, at least, the right column (rows are shorted inside a column), so the signal that was being received by the target pixel could also be observed by the oscilloscope. With the oscilloscope, one point on each signal was selected to have a reference on how they shifted. These two points were set at 85% of the amplitude of the pixel output signal and 65% of the incoming injection signal. With the measurement function of the oscilloscope, the time difference of these two points was determined for different pixels, effectively giving a relative delay that could be compared for the different pixels. How this looked is shown in Fig. 3.6. This procedure was repeated for several pixels.

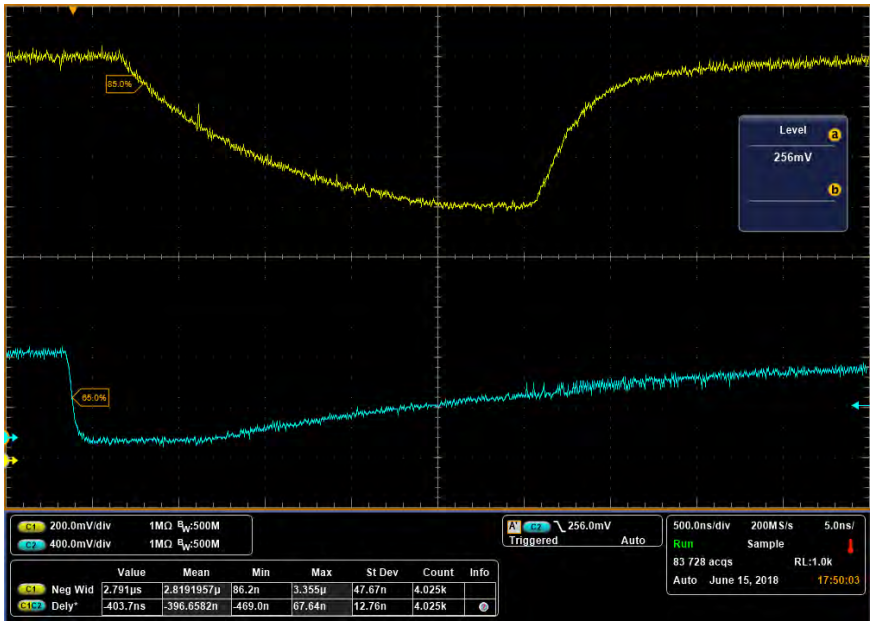

Figure 3.6.: Screenshot of the oscilloscope showing initial measurements of a relative time delay.

This measurement was done over a period of a few minutes to allow for more counts and improve statistics. The mean value of these delays was recorded for 15 different pixels (see table 3.1). Finally, these measurements were plotted in Fig. 3.7, where the coordinates are given by the position (column, row) of the pixel and the value of the delay is displayed using colour.

Even though this is a really rudimentary measurement, it shows the phenomenon pretty well. It shows a clear pattern of the delay increasing as the position gets farther from the origin.

Table 3.1.: Delay of injected and hitbus signal measured with the oscilloscope

Column	Row	Average delay [ns]	Standard deviation[ns]
0	0	323.5	16.9
1	50	369.5	12.0
1	100	382.5	30.1
1	150	366.5	22.2
1	195	419.6	32.4
20	5	353.0	15.3
24	50	396.7	12.8
24	150	441.3	21.7
46	5	447.3	12.4
46	50	439.7	19.2
46	100	457.6	18.9
46	150	461.0	20.5
46	198	448.6	17.1

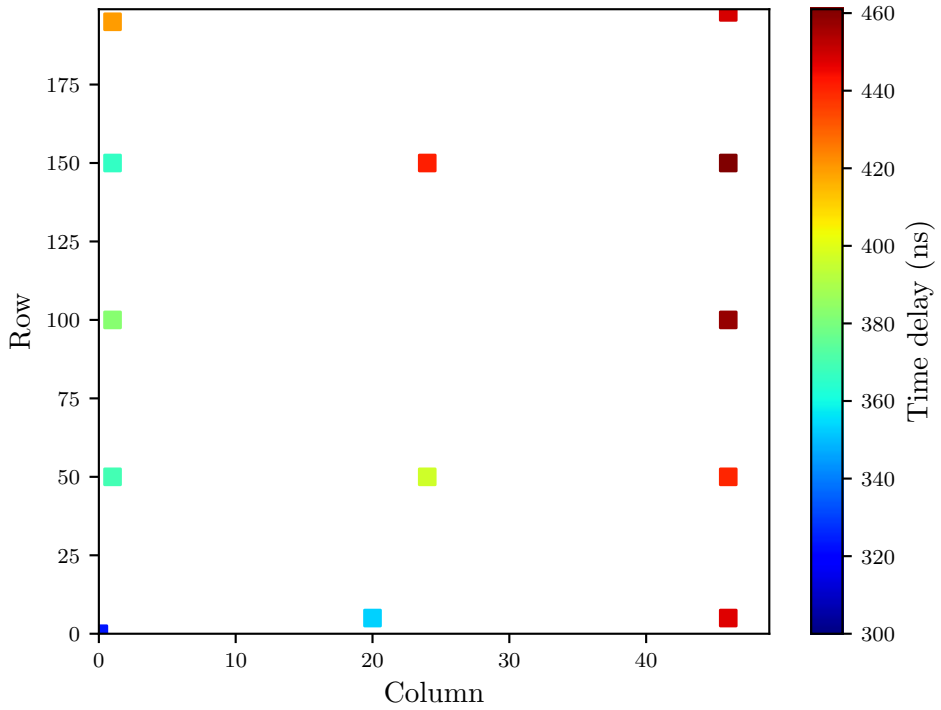

Figure 3.7.: Relative delay between injection and the hitbus for a few pixels across the chip.

Having done this first measurement, the next step would be to measure this for all pixels.

3.3. Injecting several pixels

Measuring all the pixels of the sensor can be done in two different ways. The most straightforward approach is to scan through all pixels, injecting one by one. With a total area for the first matrix of 9600 pixels, doing this by entering the address of the pixel for each entry is not reasonable. The alternative is to automate this within the software framework. So a function was written that did the following:

1. sets the injection address for one pixel;
2. reconfigures the chip;
3. starts a measurement;
4. waits some time, while measuring;
5. stops the measurement;
6. repeats for the next pixel.

The measurement itself requires some time since a sufficient amount of pulses has to be injected to have considerable statistics. Additionally, the configuring of the pixel also takes about 4 to 5 seconds. This results in at least a window of 15 seconds per pixel and 9600 pixels, giving a total of 40 hours. To avoid these long waiting periods, an alternative method had to be found.

The second possibility would be to inject several pixels at the same time. Since it is possible from the software to set an injection vector that covers the whole matrix, this seemed to be the most practical approach. On the one hand, it would drastically reduce the measuring time from about 40 hours to the time it took to measure a single pixel. On the other hand, if all injections were fired at the same time, it eliminates the need to look for a reference time, since all the events happening at the same time and it would be as simple as subtracting the timestamps to find out how they were delayed with respect to each other. But this does not work either. When the injection vector is set to cover the matrix completely, the current to generate the injection has to be distributed over the whole area. This constrains the charge with which the pixels are being injected. Why this is a problem is part of the discussion of section 3.4.1.

Still, the possibility remains to inject multiple pixels at once. In order to achieve this, it was also necessary to find out, how many pixels can be safely injected without compromising the efficiency and also leaving enough room to tune the pixel.

3.4. Threshold scans

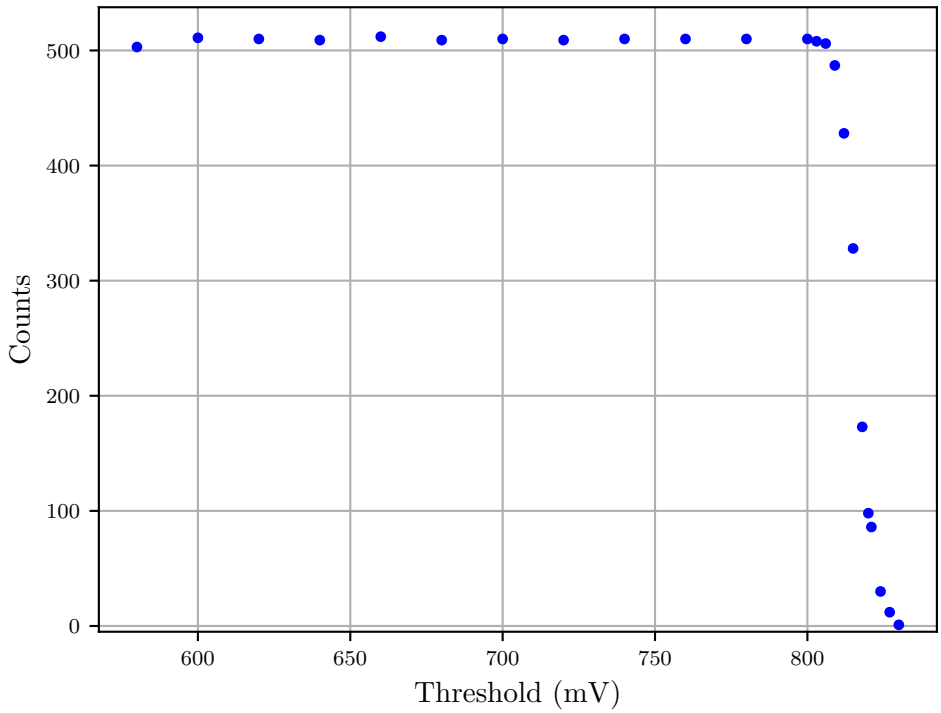

Figure 3.8.: Threshold scan of a single pixel at position (24,100)

To achieve this, it would be appropriate to measure hit counts for different values of the threshold while expecting the same number of hits. To measure counts for different values of the threshold, it is important that the source of the counts is constant and reliable. Again, the best and simplest solution is to use the injections. For each injection, the frequency and the amplitude of each pulse can be set, so there is a clear value to expect for the number of counts and also when the counts should drastically fall. This simple idea comes with several challenges, mainly how to store the measurements and how to program the scans.

Initially, the scans were very simple. Take a single pixel (configure the injection address) and start a measurement, inject it with a given frequency and amplitude (almost always $f = 100\,\mathrm{Hz}$ and $A = 750\,\mathrm{mV}$), wait for a couple of seconds, stop and store the measurement, increase the value of the threshold by a certain amount and repeat. The first problem was, that at some point, the threshold value will be so high that no events are going to be detected. Because of the way the files were written in the program, this would always cause the program to crash. This was fixed and the

program was now capable of storing files were no events happened. Afterwards, the measurement of a single scan would be saved across several files, each containing the information of a single step of the scan. But doing constant large steps like this is not helpful either. The counts remain constant for low threshold values until a point is reached where the counts sharply drop. Measuring this way means having several measurement points with high counts and on the other side several points with zero hits and nothing in between. Finding the region where the counts dropped, meant analysing the initial measurement and repeat the process for smaller steps around the area between the last point with high counts and first with zero counts.

To eliminate the need of having to do a manual analysis between the two measurements, an algorithm was developed. Essentially, the scan would work in the same way as it already did, but after finishing up each step, the measurement file (block file) would be opened and analysed. The analysis consisted of a simple hit count for the one pixel that was being injected, ignoring all others, suppressing any possible unwanted events. This number was stored in a variable which at the end of each iteration (or step) would become the value of another variable storing the number of counts of the previous step. The program would decide if the next step should have been a smaller one if the difference between the previous counts and the current counts is larger than a certain criterion (e.g. 30% of the initially expected counts). Since there is only one region where the values change suddenly, once this point was reached, a variable will switch (from false to true), which in turn lets the program only do small steps. The program continues scanning in small steps until the counts fall below a previously set limit (e.g. 10 hits), then switch the boolean variable back to false. All this process is constrained on both sides by the starting and ending values of the threshold scan.

This process is described again graphically on Fig. 3.9. The variable *on_slope* is initially set to false. The first value of *previous counts*, before the first measurement, is estimated from the time taken for the measurement multiplied with the frequency of the injections. A is the smallest value that is no longer tolerable for a jump of counts between steps. The value of B is the lower limit. The value of *upper limit* refers to the upper limit of the threshold scan (this was almost always set to 850 mV).

This function was developed further. The later features include the capability of setting injections to any desired area of pixels, while still counting a single pixel within this area or counting all pixels, with just a few modifications of the original code. At the core of this function, sits the counter. In the early stages, the counts were done reading block files, which were written on each step of the scan. A typical scan would have about 35 steps. Since a block file contains much more information than just the number of hits for one or several pixels, this is just wasteful programming. The final version of this function has all the capabilities of the previous versions but does the counts on the fly. This means that there are no extra files being written for every step but also that there is no clear separation between steps. Until this point, beginning a measurement meant creating a new clean file and just writing the new events into it.

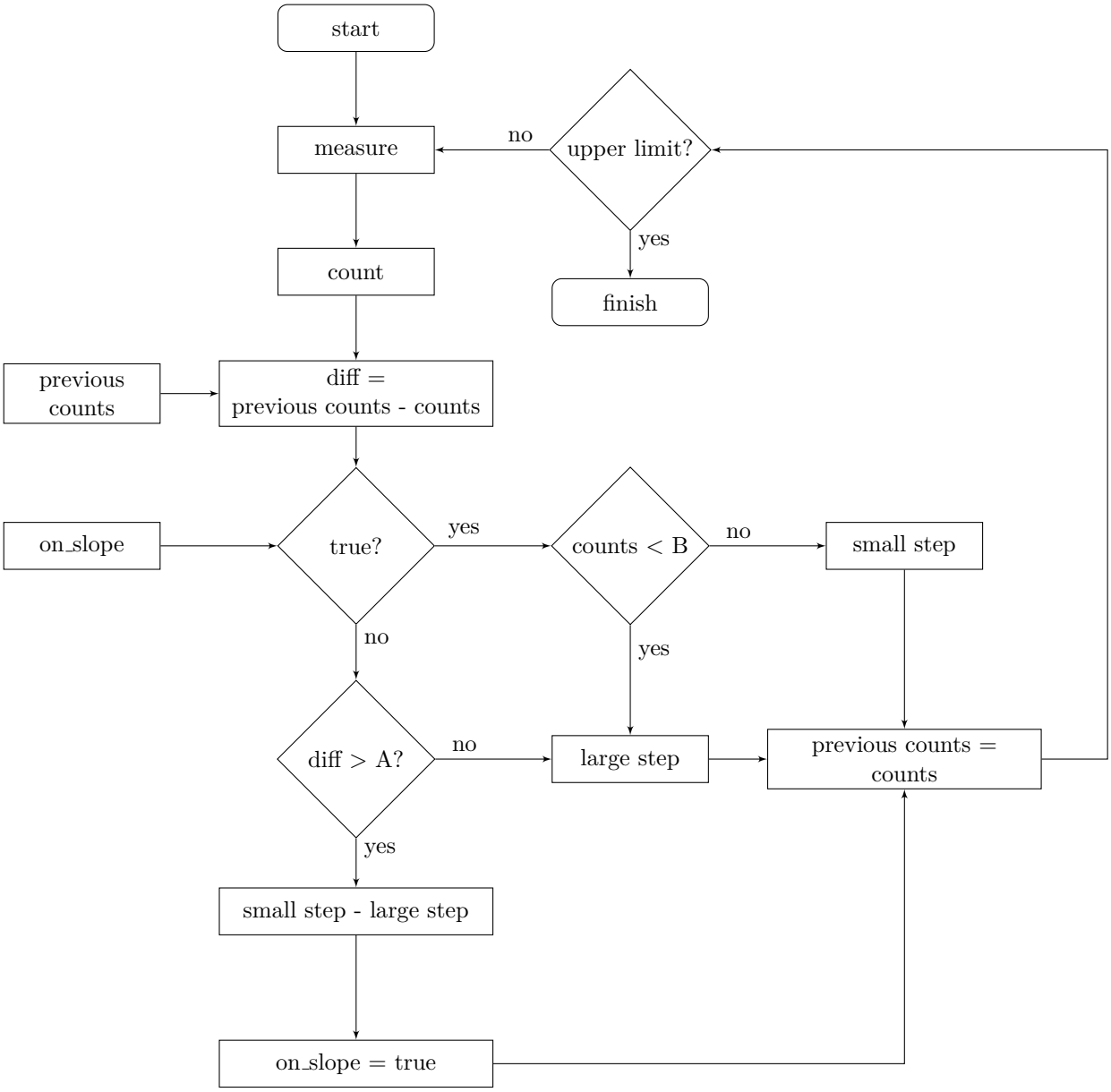

Figure 3.9.: Flowchart of the initial algorithm of the threshold scans. The variable *on_slope* is a boolean which is initially false and becomes true when the area of rapid change has been reached. A is a number above the largest difference that is accepted before one has to assume that the slope was skipped. B is a certain minimum of counts, i.e. the counts expected for the second plateau, which tells the program that the area of the slope has been abandoned and large steps can be taken again. *upper limit* is the upper bound for the threshold. Since the scan starts at the plateau, at the beginning the way to read this flowchart correctly would be to go once through answering *no* to every question.

For the current version, the program does not start any new measurements (or runs) so the hits are not neatly separated. Still, this just means that the two variables for the counts (*counts* and *previous counts*) are not enough and that two more similar variables must be defined: *total counts* and *total previous counts*. These had to be used to find how many counts there were in any given step, while the other two still participated in the logic as explained before.

3.4.1. Evaluating the threshold scans

The original purpose of these threshold scans was to find out the largest area of pixels that could be injected at once, to minimize the time required for measuring. This would mean doing a fit and comparing measurements of injections of areas of different sizes. The distribution of the points is a threshold-like function (see Fig. 3.8), with the characteristic S shape of a sigmoid curve (also simply referred to as *s-curve*). Therefore, a scaled, displaced, weighted and flipped sigmoid curve can be used to fit this data [10]. The explicit form is:

$$f(U) = \frac{A}{1 + \exp\left(B\left(U - C\right)\right)}. \tag{3.1}$$

Here $f(U)$ is used to determine the number of counts for any given voltage setting. The variables A, B and C in (3.1) are the highest number of counts, a factor that describes how step-like the function is (the higher the value the more step-like the function) and where the counts have fallen to half of the maximal value, respectively.

To evaluate different s-curves for injected areas of different sizes, the most important factor is C. The measurements show that injecting a larger area displace the whole curve to the left. This means that the counts fall for lower values of the threshold. This displacement can be generalised as the variation of C. In other words, C is a measurement of how efficient the pixel or the area of pixel is. The value of B can be used to estimate the noise. To gain more information about this relation between the number of pixels being injected and the drop in efficiency, several threshold scans were recorded.

As already mentioned, there are two possibilities for counting. One could count by picking one single pixel of the injected area. The alternative would be to count the hits of all pixels in the injected area. The latter may offer better statistics, since not all pixels are equally sensitive. For the purpose of finding out, how large the injection area can be set, the second method is better suited to estimate the pixel to pixel variations.

Two sets of measurements were taken, one doing single counts and one doing area counts. The measured values were recorded in figures (see sections A.1 and A.2). In order to do a more quantitative analysis the measurements were fitted. The estimated fit values can be found on tables 3.2 and 3.3.

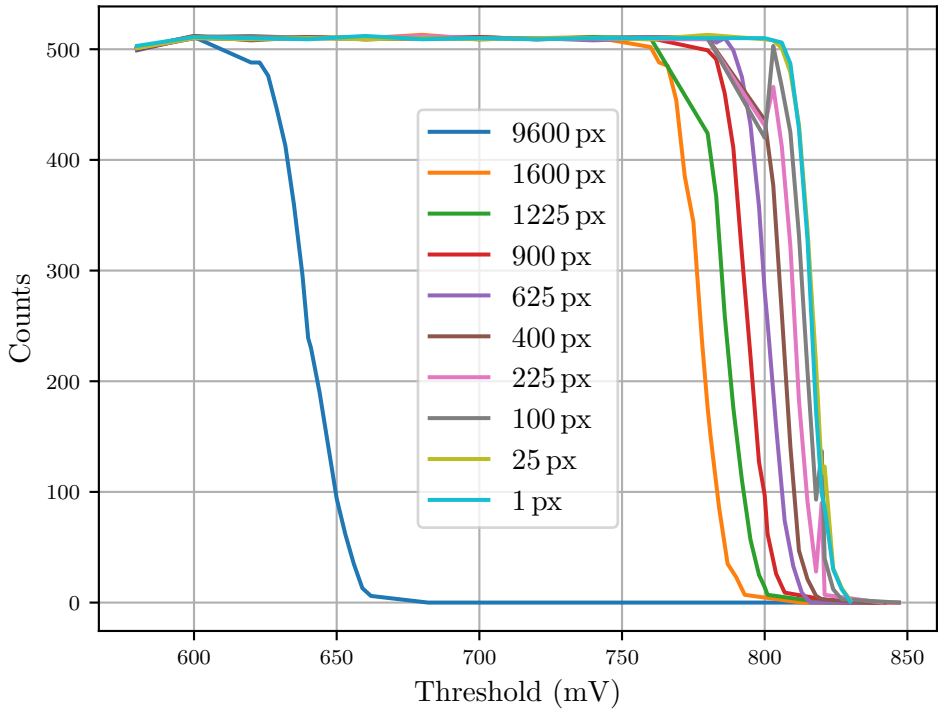

Figure 3.10.: Comparison plot of several threshold scans with different injected area sizes. For all scans, the counts were done on the pixel (24,100).

Table 3.2.: Fit values of the s-curves for different amount of pixels. The counts were taken from a single pixel at (24,100).

Pixels	A	B [mV^{-1}]	C [mV]	χ^2
1	510.175 ± 1.150	0.384 ± 0.007	816.429 ± 0.057	0.601
25	509.475 ± 1.864	0.347 ± 0.010	816.991 ± 0.095	0.553
100	505.029 ± 6.946	0.295 ± 0.028	814.090 ± 0.393	2.671
225	506.367 ± 6.297	0.298 ± 0.026	810.449 ± 0.350	6.610
400	508.580 ± 1.657	0.338 ± 0.009	806.026 ± 0.084	1.267
625	510.048 ± 1.189	0.289 ± 0.005	800.936 ± 0.063	2.170
900	509.441 ± 1.470	0.274 ± 0.005	794.038 ± 0.082	2.195
1225	509.654 ± 1.492	0.249 ± 0.005	786.468 ± 0.086	2.272
1600	508.962 ± 1.936	0.246 ± 0.006	777.448 ± 0.107	2.176
9600	511.115 ± 4.565	0.163 ± 0.005	640.238 ± 0.213	9.567

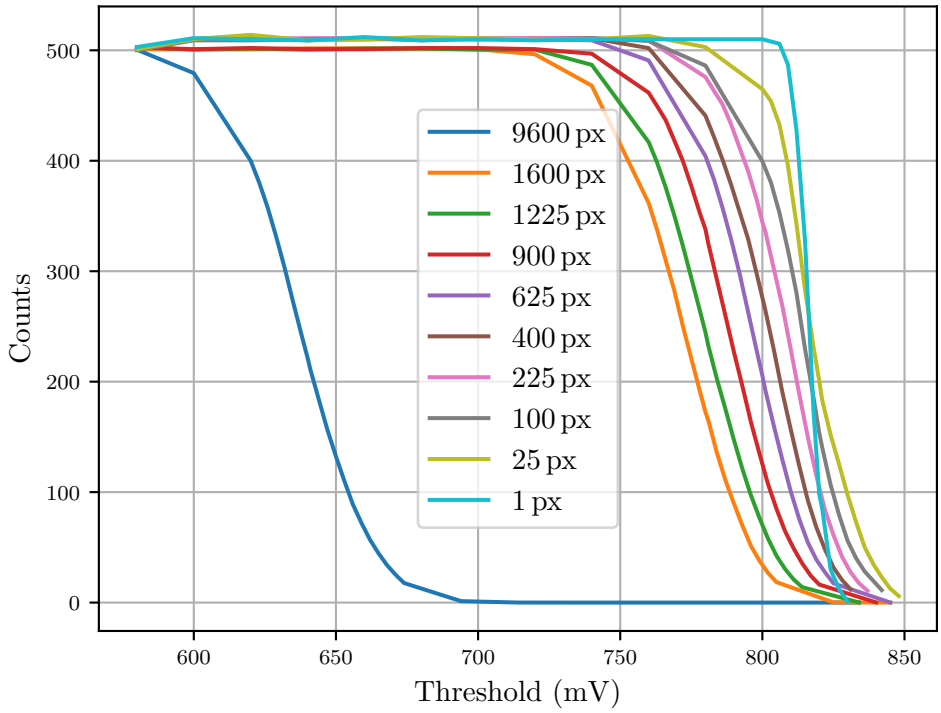

Figure 3.11.: Comparison plot of several threshold scans with different injected area sizes. For this, the counts were done using all pixels. Furthermore, the counts were divided by the area as a kind of normalisation.

Table 3.3.: Fit values of the s-curves for different amount of pixels. The counts were taking from the whole injected area.

Pixels	A	B [mV^{-1}]	C [mV]	χ^2
25	12784.147 ± 64.921	0.130 ± 0.003	817.394 ± 0.216	0.348
100	50707.060 ± 189.056	0.117 ± 0.002	813.020 ± 0.173	0.163
225	114154.491 ± 501.875	0.104 ± 0.002	807.278 ± 0.216	0.590
400	203110.934 ± 1062.776	0.097 ± 0.002	801.111 ± 0.258	0.615
625	317659.952 ± 1137.889	0.096 ± 0.002	795.421 ± 0.179	1.241
900	450679.807 ± 1071.177	0.091 ± 0.001	787.656 ± 0.121	1.156
1225	613958.170 ± 1802.365	0.086 ± 0.001	779.172 ± 0.150	1.282
1600	801443.600 ± 2462.747	0.085 ± 0.001	771.958 ± 0.166	2.151
9600	$4827140.591 \pm 16540.030$	0.082 ± 0.001	637.038 ± 0.121	7.479

Fig. 3.10 and Fig. 3.11 show the expected behaviour. The measurement curves shift to the left with the increasing number of pixels. In this case, the shift comes not from a drop of the pixel efficiency but from the decrease of the injection amplitude.

As mentioned in section 2.3.3, the injection is basically done by a large capacity that charges another smaller capacity. In the case of a single pixel, the difference in capacities is so large that for the pixel, the charge of the injection capacity acts as a constant voltage source. With an increasing number of pixels, the total capacity that has to be charged increases. Because of this, the difference drops. The result is that the amplitude of the injection decreases for every pixel.

Finally, the displacement of any s-curve can be explicitly calculated, if one defines the C value for one pixel as reference. This means the displacement is given by the difference between the reference value and the values for any given amount of pixels. The calculated values, using the C values from table 3.2, can be seen in Fig. 3.12.

The resulting displacement were plotted for the first nine measurements using a linear plot. The fit result for the slope of the linear function $m = (0.0252 \pm 0.0003)\,\mathrm{mV/px}$ ($\chi^2 = 3.79$). As already mentioned, for a small collection of pixels, the difference of the capacities is so large that the voltage coming from the injection is almost a constant voltage source. This is reflected in Fig. 3.12 for the linear region. For a much larger area of pixels, the total capacity to be injected is comparable to the injection capacity, and the behaviour is no longer linear.

Finally, it should be noted that for the area counts, the value of B changes more drastically than for the single counts. This would normally indicate a noisier measurement. But if these measurements were done again, for any other single pixel, they would most likely have a similar shape to the one seen in Fig. 3.10. The decrease in the value of B comes, in this case, from the overlapping of s-curve from all pixels in the injected area. Even if these all have a sharp step, the superposition of the small variation of the C values, generate a flatter s-curve.

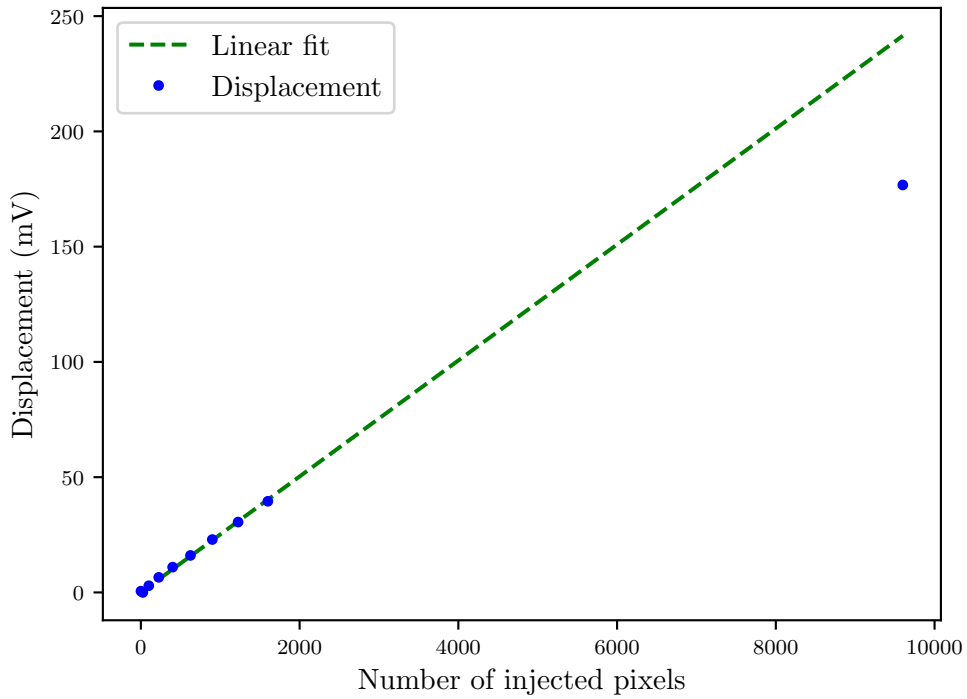

Figure 3.12.: Relative displacement of the s-curves when injecting different amounts of pixels. The C values needed to calculate the displacement were taken from table 3.2. A dash line shows a linear fit of the first nine measurement points.

3.4.2. Alternative method

As an alternative to doing a complete threshold scan and fitting, a binary search would also result in determining for which voltage the counts have dropped to half of the initial value. This can only be done, because the expected values of the counts are known. In this case, while injecting, the counts that are expected are given by the product between the number of pixels being injected, the time set for each step of the scan and the frequency of the injections.

For the binary search to work, the threshold voltage must be set midway between to the beginning and the end of the scan, i.e. if the scan were to be between 600 mV and 900 mV, the initial value would be set to 750 mV. At this voltage, the program would measure for a given time and return a number of counts. If the counts are over the half of the initially expected value, then the voltage is set to value midway between the current value and the end point. Otherwise, if the number of counts is

smaller than half of the initial value, then the next voltage setting is halfway between the lower limit and the current value. By redefining the upper and lower limits like this, the range where the searched value becomes increasingly smaller. Eventually, the voltage for which the counts have fallen to half is found. A graphical description of this process is given on Fig. 3.13.

Even though this process might be quicker in finding the displacement of the whole curve, it does not provide other pieces of valuable information like the other method does. At the end, it becomes a matter of what information is needed. To simply measure how the s-curves are displaced, this becomes the better alternative, because it will most likely end needing less measurement points and not having to fit the measurements. But by reducing the amount of measurements, the information about the noise of the pixel (or area of pixels) is also lost.

Because of the limited time available for this thesis, this method was not implemented.

3.5. The time delay

At this point, the expectation was that any pixel would have a relative delay and that it could be altered by changing the threshold voltage. Figuring out how many pixels could be injected at once was a matter of understanding how much the threshold for a single pixel would have to be modified. Ideally, for any given area of pixels, there would be enough room to change the threshold to correct, or at least improve, the relative delay while remaining on the first plateau.

The next step would be to figure out how to measure the time delay. Due to the fact that this is a relative measurement, it is necessary to define a reference. Since the idea is to inject several pixels at once, the logical reference would be the time at which the injection was sent. The time delay is then define as the difference between the injection timestamp and the hit timestamp.

3.5.1. Synchronising the timestamps

After the firmware of the FPGA was modified to store the injection timestamp as the trigger for every frame, it was still necessary to figure out how these two timestamps relate. As already discussed in section 2.4, these timestamps run on different but synchronised clocks. Additionally, for the hits, there is only 10 bits. Meanwhile, the triggers are stored using 48 bits unsigned integers. Because of the difference in the frequency, it is clear that for every four ticks of the quick clock, the slow clock ticks one time. This can be easily be synchronised by dividing the timestamps of the quick clock by four. Because of its larger range, the trigger values have to also be limited to 1024 possible values that can be taken by a hit timestamp. This is achieved by taking the modulo 1024 of the already slowed down trigger value. So the formula

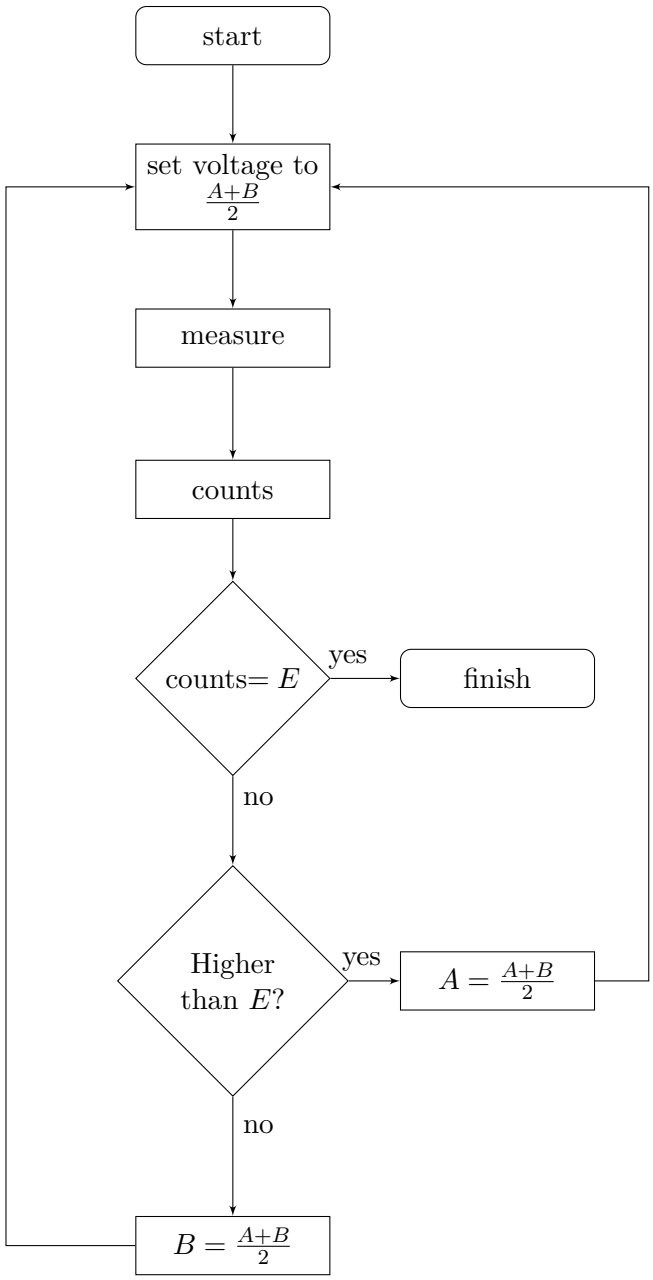

Figure 3.13.: Flowchart of the binary search method. Here E is the number of half the counts, A and B are the lower and upper limits of the search, respectively.

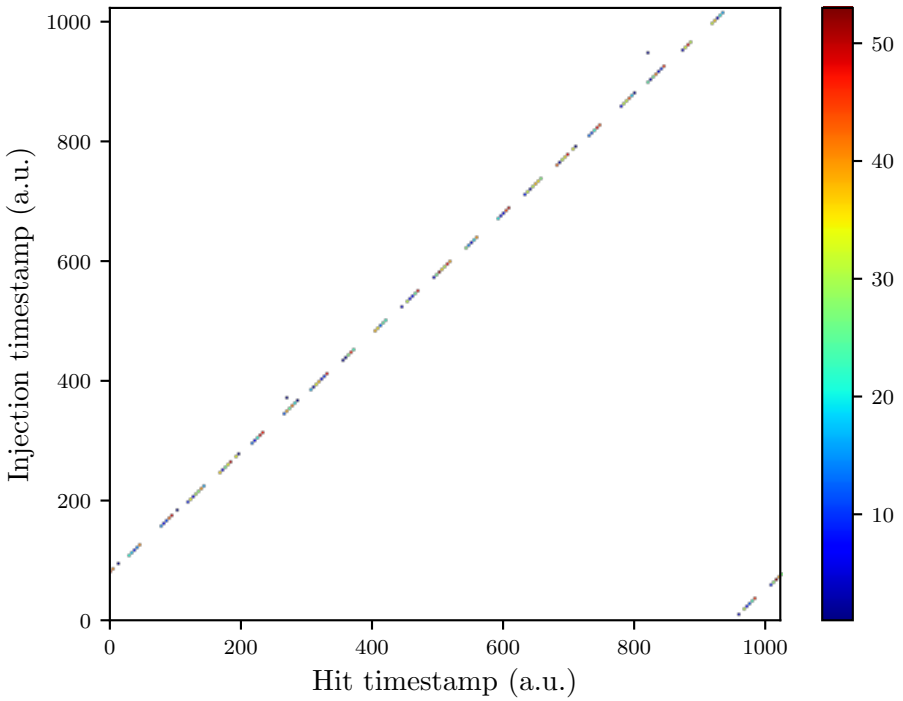

Figure 3.14.: 2D histogram showing the injection and the hit timestamp for every measured hit. The colour bar indicates how many hits had the same injection and hit timestamp.

to obtain an useful injection timestamp from the value stored in the trigger variable is:

$$t_{inj} = \left(\frac{t_{trigger}}{4}\right) \bmod 1024. \tag{3.2}$$

The mod in (3.2) returns the remainder obtained when dividing the number before it by the number behind it (e.g. $10 \bmod 3 = 1$, since the remainder of $\frac{10}{3}$ is one). In this case, it returns exactly the value of injection timestamp, making sure it always resets when the number reaches 1023. Having done this, the value of t_{inj} should have the same properties as the hit timestamp.

To verify that the two timestamps were in fact synchronised, a measurement of a single pixel was taken. The 2D histogram in Fig. 3.14 shows every measurement of both types of timestamp. This measurement contained around 54,000 hits and was taken on a single pixel at (15,100).

In the ideal case, Fig. 3.14 should be just one straight line. But since some delay is expected, there has to be an offset between the injection and the hit. In the specific case of this measurement, this delay is just below 1000 (it will still be estimated precisely in the next section). Nevertheless, this measurement still proves that the timestamps are now synchronised, because the line is a diagonal with a slope equal to 1.0

The values on both axes are in the range between 0 and 1023 and are of the units of a timestamp. This measurement confirms that the frequency for both sets of timestamps is 125 MHz. With a known frequency, the corresponding time for each timestamp unit can be determined to be 8 ns. From this point onward, the time delays will be given in ns.

3.5.2. Measurements of the time delay on a single pixel

With a well defined reference, the delay can be calculated for any given pixel. The time delay is that explicitly defined as:

$$t^*_{delay} = (t_{hit} - t_{injection}) \mod 1024, \tag{3.3}$$

where t^*_{delay} is the difference between the hit and injection timestamp. Normally, the t_{hit} should always be greater than $t_{injection}$, but when the former reaches 1023 the counter returns to 0, while the $t_{injection}$ is still getting larger. The result of this difference would be a negative value. To correct this, the modulo is applied. This ensures that the value obtained is always just the delay between the injection and the hit signal. Finally, (3.3) is multiplied by 8 ns to obtain the time delay as a physical time. The delay expressed in ns is from now on noted as t_{delay}.

By inputting all t_{delay} from every injected pixels in separate histograms and fitting them, allows to find the best estimate of the individual delays. Fig 3.15 shows a typical measurement for a given pixel. With the exception of a small amount of outliers, most pixels show a similar distributions. The outliers come from hits that are stored in a frame without injection timestamp. No further corrections were applied here, because these very few outliers do not affect the statistics considerably. The usual width of the peak in the histogram is normally no more than 3 bins.

3.5.3. Measurements of the time delay on a small area of pixels

After having developed a method to measure the time delay for any given pixel, the next step was to inject areas of several pixels. At some point, the whole matrix should be measured. Doing this block by block, seemed to be the quickest and most practical way to measure.

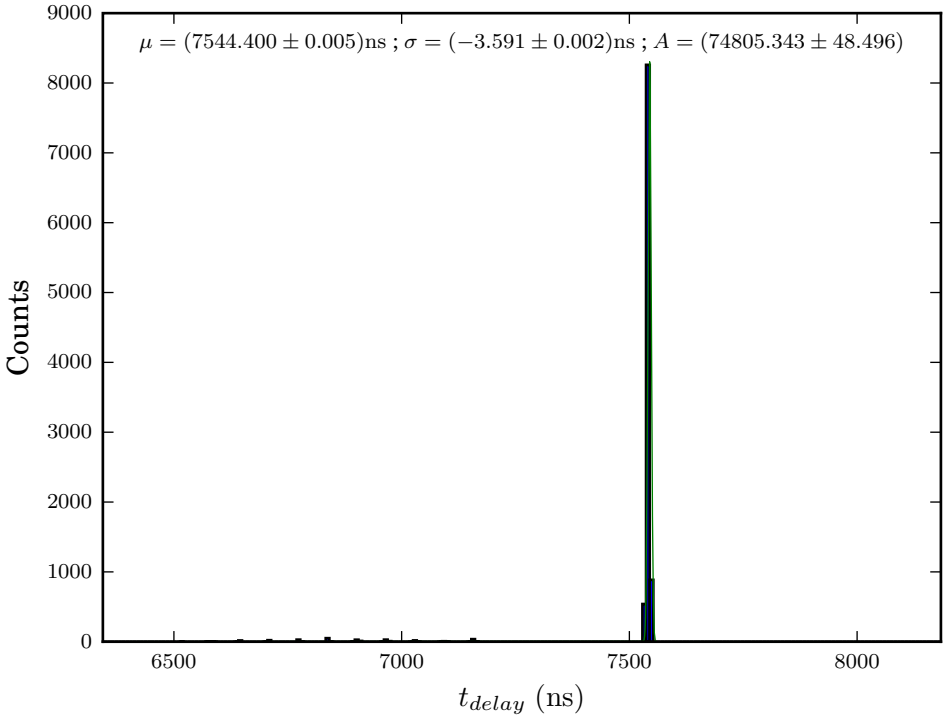

Figure 3.15.: Histogram and fit for the delay of a single pixel at (33,13).

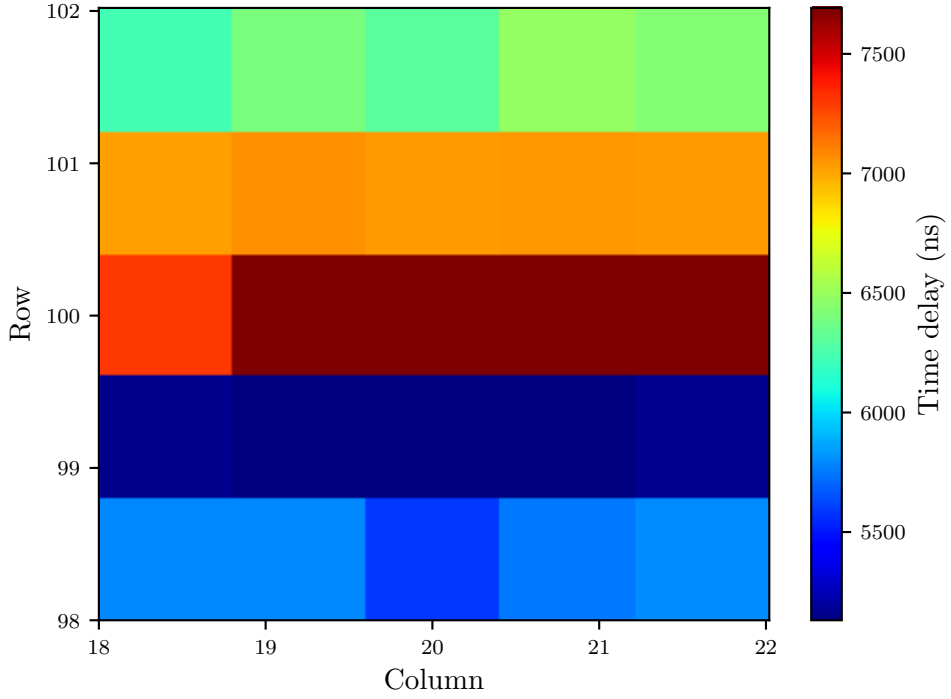

Figure 3.16.: Time delay measurement of a small area being injected simultaneously. The results shown here do not correspond to the actual delays. This is shown to explain a systematic error that comes from measuring several rows at the same time.

As a first try, a small area of 25 px was injected and the hits recorded. This amount of pixels is still in the linear portion, and according to the fit of section 3.4.1, the displacement of the s-curve should be $\Delta U = (0.6300 \pm 0.0075)\,\mathrm{mV}$. Given that this could vary from pixel to pixel, this variation is not large enough to affect the results of the individual delays. The results of this measurement are shown in Fig. 3.16.

By looking at Fig. 3.16, it may seem as the value of the delay changed strongly between the rows and remained relatively constant inside a given row. A more careful analysis of the data showed that the difference between rows is actually an effect of the read out. As described in section 2.3.1, when several pixels inside a column register a hit, there is a certain hierarchy. In this case, the information of a single row is written into a telescope frame, while the rest has not been read out yet. Eventually, every hit is read out, but every row is filled in different telescope frames.

The problem is that the trigger, or in this case the injection timestamp, is stored in

the telescope frame. And since there is only one injection for all 25 pixels, the frames that follow do not have a proper injection timestamp. A closer look at the delay values showed that in this case, the hits from row 100 had the trigger in their frame and had the expected values for the delay. After that, the value stored as the injection timestamp increased by 640 ns (namely, in the following row order: 100,101,102,98,99).

This effect is purely systematic. Still, considering the short time available for this thesis, the decision was made to not try to measure the delay from whole areas and rather measure single rows. Complete rows can be measured at the same time, so that for any injection all the corresponding hits are stored together in one frame. Also, if that is the case, the method described in section 3.5.2 could be easily adapted.

3.5.4. Measurements of the time delay on a single row of pixels

With a few changes of the code, it was now possible to configure the injections to all columns of a given row at the same time. After that, measuring the delay for all pixels could be done, as it was already described in previous sections for every pixel in the row.

This measurement was repeated with the same configurations to test how consistent they are. The results can be found in the appendix under section A.3. It should be noted, that with the same configuration, the relative delay of each pixel remains almost identical, which is reflected in the colour value for every pixel.

As rough estimate of the time resolution of a single row, the difference between the maximum and minimum delay was calculated. This method is not ideal, since it relays on single pixels to estimate a value that could vary strongly for a given pixel, but seen as larger group remain more homogeneous (with a handful of outliers). The correct way to calculate the time resolution for a row (or any number of pixels in general) is done exactly the same way it should be done for a single pixel. Instead of inputting the time delay for a single pixel in a histogram, the values for all hits of a given row should be considered. The overlap of the slightly shifted histograms gives a fit Gauss curve which is logically wider. The time resolution is defined as the estimated σ parameter for the Gauss curve.

For the measurement in Fig. 3.17, the differences gives an estimate of the time resolution of $\Delta t = (50.962 \pm 0.007)$ns.

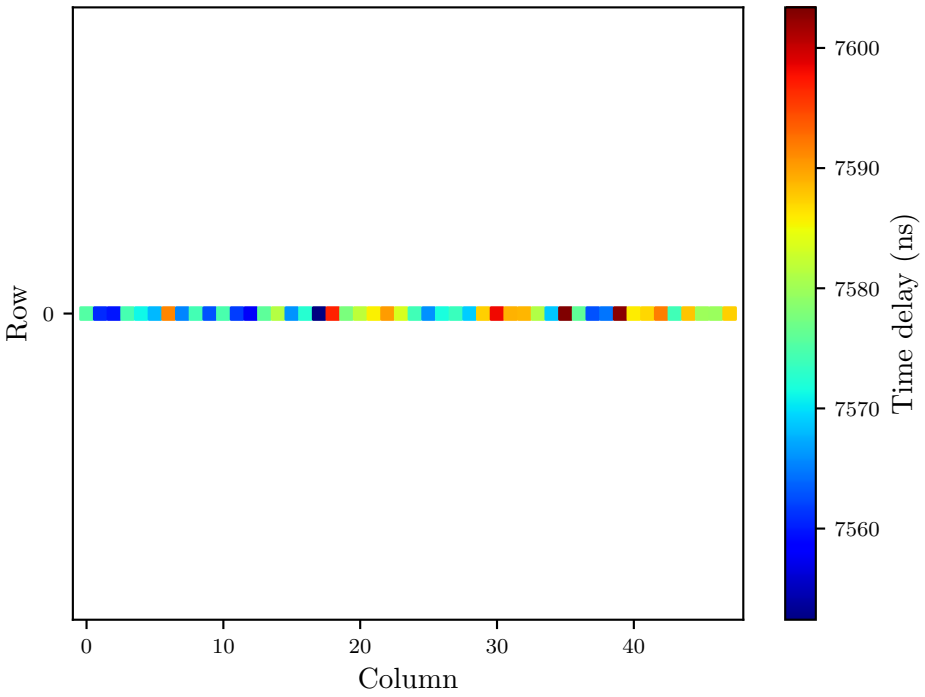

Figure 3.17.: Time delay values for all pixels on row 0.

3.5.5. Measurement of the time delay on the whole matrix

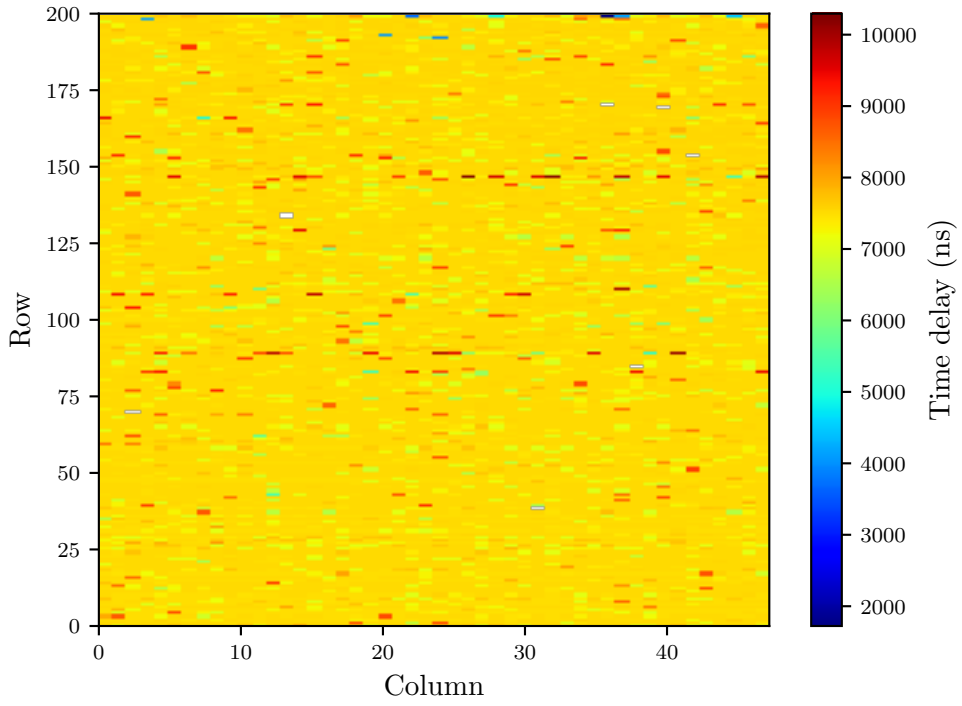

Figure 3.18.: Delay values for every pixel in matrix A.

Having successfully injected and measured the delay of a single row, a measurement was programmed to iterate through all 200 rows. Between iterations, a sleep function forces an adjustable waiting period to collect more measurements. In the case of the measurement shown in Fig. 3.18, the waiting period was set to 100 s, which with a injection frequency of 100 Hz totals 10,000 hits per pixel per row.

A waiting period of 100 s for all 200 rows is almost 5.6 h. Because of some problem that has not been determined yet, the program crashes after certain number of measurements. This made the measuring process more tedious and lengthy, because every time the program crashed, a new set of bounds has to be passed. At the end of the measurement, the program returns a text file containing two columns, one is the run number (i.e. an identifier) of the block file and the other one is the row. The analysis of this measurement is done by looping through this file, finding the right block file and the right row, and repeating the analysis as it was done for a single row. This means that the process shown Fig. 3.15 had to be repeated for almost all 9,600 pixels.

There are a certain numbers of pixels that register non-stop hits. These so called hot pixels are handled by masking. When a pixel is masked the value of threshold is adjusted so high, that the pixel is effectively turned off. For these pixels there are no measurements and the time delay can not be estimated. These are the white rectangles in Fig. 3.18.

The delay is for the most part homogeneous across the whole matrix, with most pixels registering delays of roughly 7400 ns to 7500 ns. Also many other pixels show strong deviations from these values in either direction. But contrary to the expectation (and to the measurement taken in section 3.2), the time delay does not show any position dependence. More than anything else, the delay appears to be random for most pixels. Although the density of pixels with higher delays seems to be larger in the top right quadrant and towards the center of the chip.

To obtain a better idea of the pixel to pixel variation, it is necessary to compare them not to the injections individually, but to each other. For this, the pixel at position (0,0) is defined as a reference. The new definition of the delay is given by the absolute difference of any other pixel to the reference.

Fig. 3.19 shows the pixel to pixel in more detailed. The mean value of this difference was calculated for all measured pixels: $\bar{t}_{delay} = (91.483 \pm 2.309)$ns.

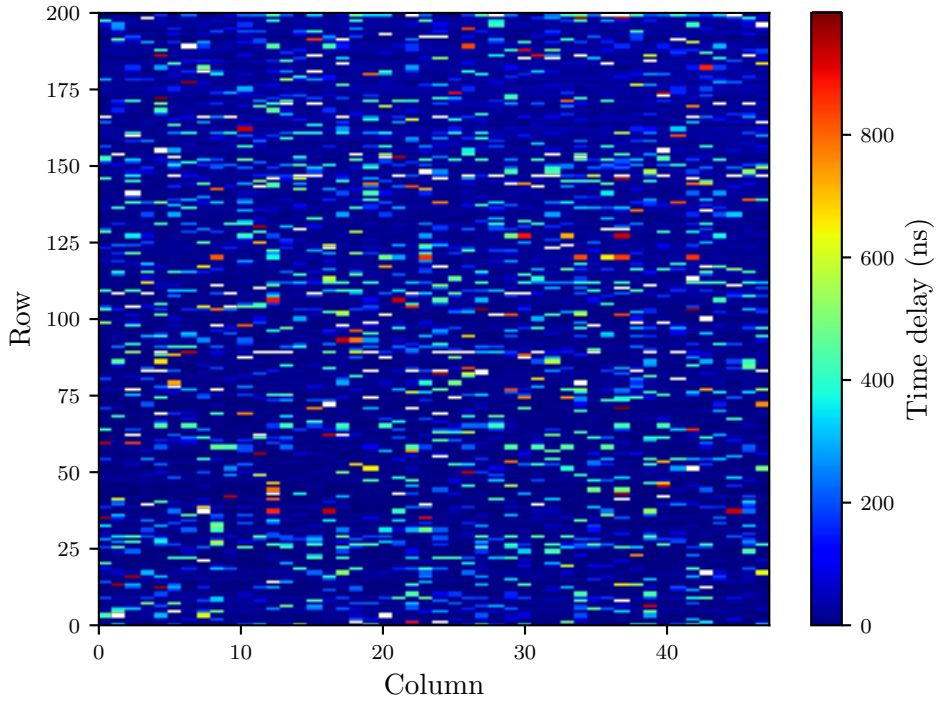

Figure 3.19.: Absolute value of the relative delay for every pixel in matrix A to pixel at (0,0).

3.6. Tuning

Once the measuring techniques had been developed and a first set of measurements had been successfully done, one could investigate how the adjustment of the threshold could modify the measured time delay.

For tuning there are three important variables. TDAC1, TDAC2 and VPDAC. The first two are variables that define an increase in set threshold of a given pixel, for U_{High} and U_{Low} respectively. VPDAC is the factor which defines how large every step of the variation of the individual thresholds is going to be, i.e. it acts by multiplying itself with the values of TDAC1 and TDAC2. If the VPDAC is set to zero, then there is not going to be any variation. The values of TDAC1 range from 0 to 7 and the values of TDAC2 range from 0 to 3. VPDAC takes values between 0 and 63.

3.6.1. Determining the range of TDAC1

As a first try, three measurement were taken to see how much the TDAC1 values can affect the time delay and the efficiency of a given pixel. For this measurements the value of VPDAC was set to its maximum. TDAC1 was set to three different values and for each setting a single pixel was injected for a certain amount of time. To test the efficiency, a threshold scan was done for each setting of TDAC1. The corresponding fits can be found in section A.4 and the fit values can be seen in table 3.5. The effect on the time delay is documented on table 3.4.

Table 3.4.: Delay values for three different setting of TDAC1. This measurement was recorded from injections set to pixel (15,15).

TDAC1	μ [ns]	$\Delta\mu$ [ns]
0	7526.360	0.008
4	7543.720	0.008
7	7517.112	0.008

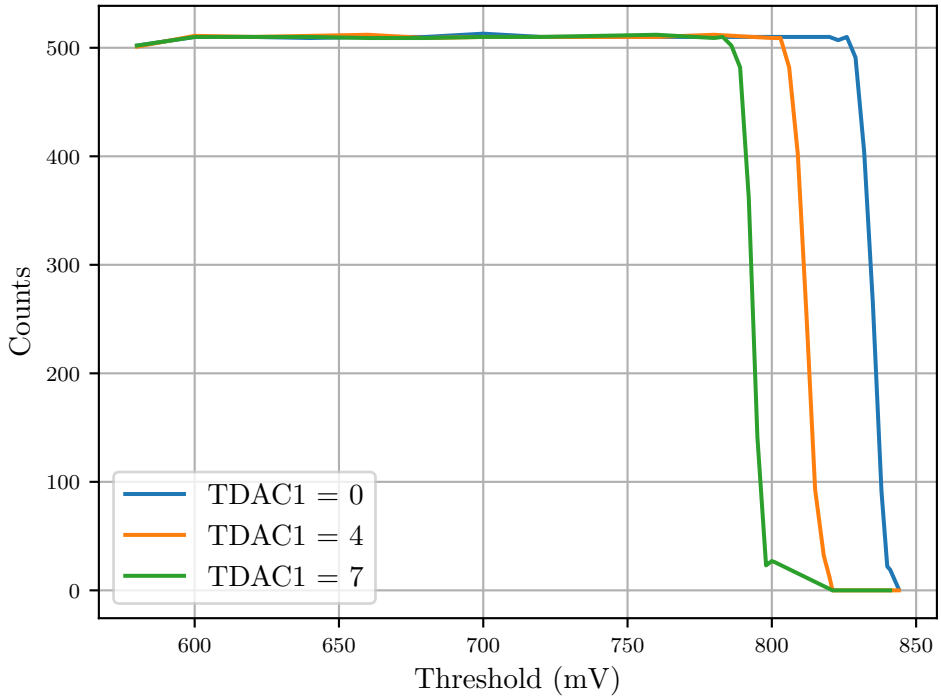

Figure 3.20.: S-curve comparison for different values of TDAC1.

Two observations can be made from these measurements. The first is that the delay is weakly influenced by the tuning of U_{High}. In fact, the small change in the time delay does not seem to be directly related to the value set for the threshold.

Table 3.5.: Fit values of s-curves for different values of TDAC1

TDAC1	A	B [mV^{-1}]	C [mV]	χ^2
0	509.574 ± 1.555	0.513 ± 0.015	834.983 ± 0.072	0.814
4	510.174 ± 1.038	0.471 ± 0.009	811.913 ± 0.047	3.037
7	509.540 ± 1.957	0.610 ± 0.024	793.462 ± 0.078	22.498

The second one is that efficiency is much more sensitive. It is apparent how the s-curves are shifted for the non-zero values of TDAC1. Under the assumption that this behaviour can be generalized for any other pixel, this measurement would also allow to calculate what the combination of VPDAC=63 and TDAC1=7 is equal to in mV, and thus the range of the tuning values. Considering that TDAC1=0 represents no shift of the s-curve while TDAC1=7 represent the maximum shift possible, the difference between the two C for the respective tune values, should result in good estimate of the range of the tune values. By doing this, the range of the tune values is estimated to be (41.521 ± 0.150)mV.

3.6.2. Methods to change the time delay

The fact that the time delay was so not changed even when applying the maximal tune value was unexpected. Trying to understand why this happened, led to trying different methods just to see if it was possible to change this value by adjusting the threshold somehow. The first idea was supported by the fact that the change in the threshold did not cause time walk. This must mean that the leading edge of the signal was so steep that it did not make much difference for the timing. Following this train of thought several more measurements were done with lower injection voltages.

Initially, the value of the injection was set to be just above of the threshold. This did not make any difference in the time delay either. Soon after, it became clear that the value that is set in the software for the injection does not correspond to the signal being injected to the pixel, so the pulse being injected was still too high and steep. After that, the voltage of the injection was configured in such a manner that it was barely above of the point where the counts began to fall. This could have been more quantitatively by determining the signal height via the threshold scans, but was not implemented because of the limited time in the scope of this project.

Table 3.6.: Time delay at the edge of the plateau with injection at pixel (30,130) with a moderate amplitude.

TDAC1	μ [ns]	$\Delta\mu$ [ns]
0	7604.264	0.008
1	7619.464	0.008
2	7617.856	0.008
3	7618.536	0.008
4	7645.672	0.008
5	7630.872	0.008
6	7664.288	0.008

Measuring this way did show a change in the signal. By setting the real injection amplitude to a value just above the threshold, increasing the tune values caused the counts to drop and the distribution of the time delays to get wider. The fits used to determine the time delay did show other values for the time delay but also showed a drop in the resolution. This is counter-productive.

Finally, having calculated the range of the tune values before, the decision was made to measure again using a small injection and setting the high threshold just below the real injection voltage by 43 mV. The idea was to see what would happen to the time delay while measuring an injection amplitude about 150 mV above the 500 mV baseline, while changing the tune values at the right end of the first plateau. Just like all the other methods, this does not show any significant changes in the values of the time delay. The measured values can be seen in table 3.6.

4. Conclusions and outlook

During the development of this project, the main goal was to study and try to improve the time resolution of the sensor by using the tuning available on the sensor. To achieve this, new software functions and analysis tools had to be implemented.

The first development were the automatic threshold scans. This function allows to set the starting and ending thresholds of the scan, the time per step (this combined with the injection frequency gives the number of injections per step) and the size of two different kinds of step (big steps for the plateau and small steps for the slope of the s-curve). The program is also capable of setting the injection vector either to a single pixel or an area of pixels. It also lets the user choose from two possible methods to do counts while scanning the threshold. The output of the scan is a single file that contains the all the voltage values that were set during the scan and the corresponding number of counts. Although, the threshold scans were never implemented as originally intended (because of the problem described in section 3.5.3), they still are extremely helpful.

Even inside this same project, the threshold scans proved to be useful in ways that were not directly related to the reason why they were implemented in the first place. They were used to find out how the tune settings translate to the adjustment of the voltage. Furthermore, the scans can be used to find out the real amplitude of a given signal. This could be especially helpful when using the injections, since the value set in the software does not correspond to the real injection.

In the scope of this project, a method for measuring the time delay between the injection and the hit timestamp was developed. If one defines a reference value, this method can be used to find out how two or more pixels are delayed in respect to each other, when recording simultaneous events. This is of great relevance since the problem of the time resolution still needs to be solved. Based on this new functionality, a new feature of the *single* software was implemented to measure the time delay for the whole submatrix A of the sensor.

There is still room for improvement for these new features. The threshold scans and the method for measuring the time delay work well, but they rely on the *sleep* command that freezes the whole graphical interface. A possible solution for this problem would be to implement a dedicated thread, were the waiting period does affect the graphical interface. Very often during the measurements, the software would cause the program or even the computer to crash. The problem remained until the end of

this project unresolved.

For tuning, none of the here applied methods could change the difference significantly between the timestamp of the hit and of the injection. Nonetheless, there are many other possibilities that could not be tested in the scope of this thesis (like using the two threshold mode and injection of moderated amplitudes while tuning the second threshold). But the fact that the regular threshold did so little to modify the timing, points towards the fact that the time delay comes not from time walk and it is not dependent on the amplitude of the signal. The real cause of the time delay will have to be found with other tests done in the future.

Another important conclusion from the attempts to tune is that the range offered to modify the threshold individually is very limited. This is a feature that will have to be improved in future prototypes of the MuPix series.

To conclude, the methods developed and the measurements conducted in the course of this thesis revealed a lot of insight in the timing behaviour of the MuPix8. The aim of the levelling the time delays of the pixel per pixel threshold turned out to be unpromising when using injections. This motivates further investigations of this effect using the described measurement techniques by using other signal sources like radioactive sources or particle beams.

Bibliography

[1] "Festkörperphysik," in *Physik für Ingenieure*, P. Dobrinski, G. Krakau, and A. Vogel, Eds. Vieweg+Teubner, pp. 574–650. [Online]. Available: https://doi.org/10.1007/978-3-8351-9076-4_8

[2] G. Lutz, *Semiconductor radiation detectors: device physics*, 1st ed. Springer Berlin, OCLC: 255964092.

[3] I. Perić, "A novel monolithic pixelated particle detector implemented in high-voltage CMOS technology," vol. 582, no. 3, pp. 876–885. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900207015914

[4] N. Berger, H. Augustin, S. Bachmann, M. Kiehn, I. Perić, A.-K. Perrevoort, R. Philipp, A. Schöning, K. Stumpf, D. Wiedner, B. Windelband, and M. Zimmermann, "A tracker for the mu3e experiment based on high-voltage monolithic active pixel sensors," vol. 732, pp. 61–65. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016890021300613X

[5] H. Augustin, N. Berger, S. Dittmeier, F. Ehrler, C. Grzesik, J. Hammerich, A. Herkert, L. Huth, J. Kröger, F. M. Aeschbacher, I. Perić, M. Prathapan, R. Schimassek, A. Schöning, I. Sorokin, A. Weber, D. Wiedner, H. Zhang, and M. Zimmermann, "MuPix8 — large area monolithic HVCMOS pixel detector for the mu3e experiment." [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900218312488

[6] A.-K. Perrevoort, "Pixel and periphery and RO MuPix7." [Online]. Available: https://www.physi.uni-heidelberg.de/Forschung/he/mu3e/wiki/index.php/File:PixelandPeripheryandRO_MuPix7.png

[7] A. Weber and I. Perić, *Documentation MuPix8 (preliminary version 1)*.

[8] Dschwen, "Comparison of threshold triggering and constant fraction triggering." [Online]. Available: https://commons.wikimedia.org/wiki/File:Constant_fraction_1.svg

[9] H. Spieler, "Introduction to radiation detectors and electronics v.5. timing measurements," p. 9. [Online]. Available: http://www-physics.lbl.gov/~spieler/physics_198_notes/PDF/V-5-Timing.pdf

[10] M. Humphrys. Continuous output - the sigmoid function. [Online]. Available: https://www.computing.dcu.ie/~humphrys/Notes/Neural/sigmoid.html

A. Threshold scans

A.1. Figures of threshold scans with counts from the pixel at (24,100)

This sections contains all figures with the measurements mentioned in section 3.4.1 using the single pixel count method.

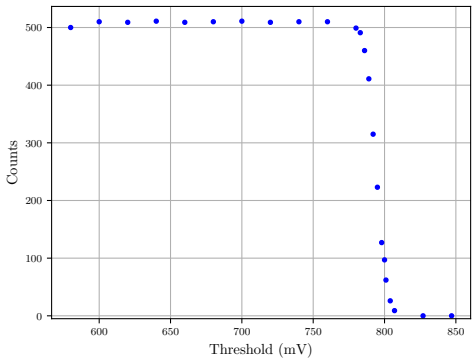

Figure A.1.: Threshold scan of the injection of 900 pixels

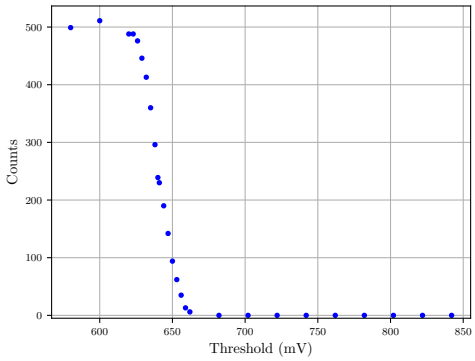

Figure A.2.: Threshold scan of the injection of 9600 pixels

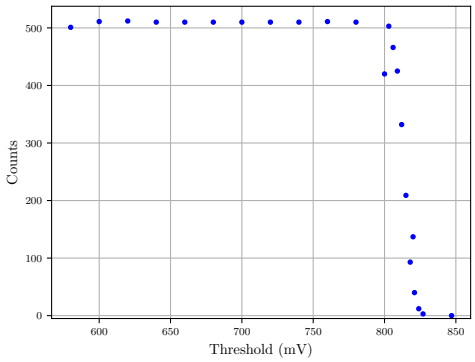

Figure A.3.: Threshold scan of the injection of 100 pixels

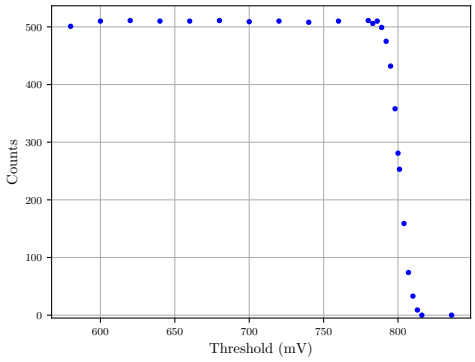

Figure A.4.: Threshold scan of the injection of 625 pixels

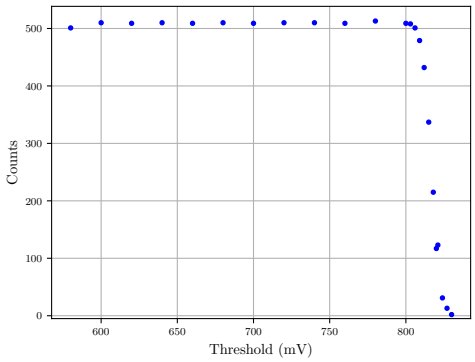

Figure A.5.: Threshold scan of the injection of 25 pixels

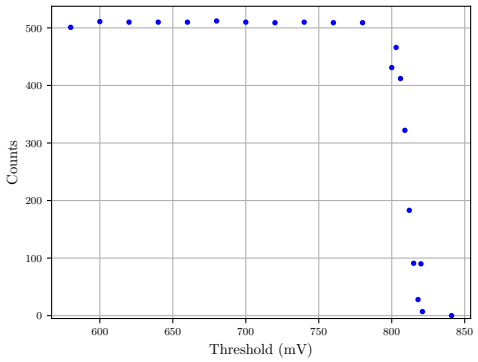

Figure A.6.: Threshold scan of the injection of 225 pixels

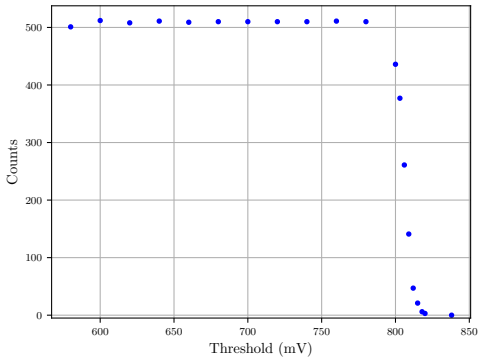

Figure A.7.: Threshold scan of the injection of 400 pixels

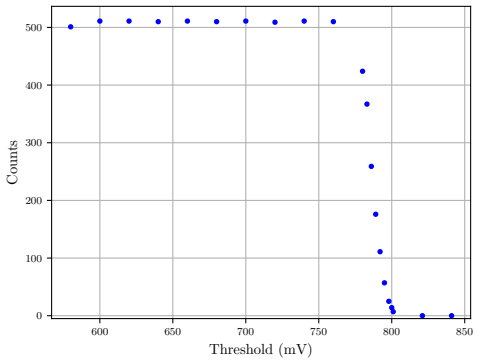

Figure A.8.: Threshold scan of the injection of 1225 pixels

A.2. Figures of the threshold scans with area counts

This section contains all figures with the measurements mentioned in section 3.4.1 using the area count method.

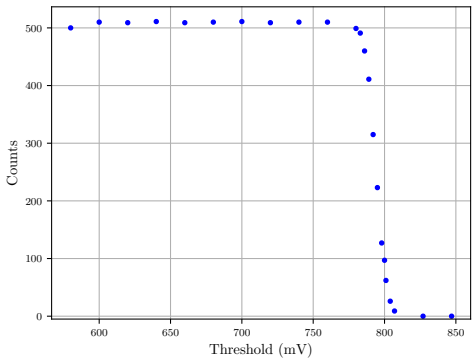

Figure A.9.: Threshold scan of the injection of 900 pixels

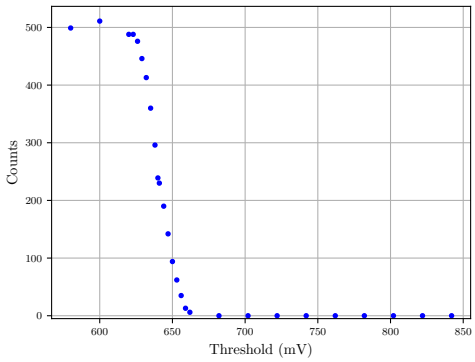

Figure A.10.: Threshold scan of the injection of 9600 pixels

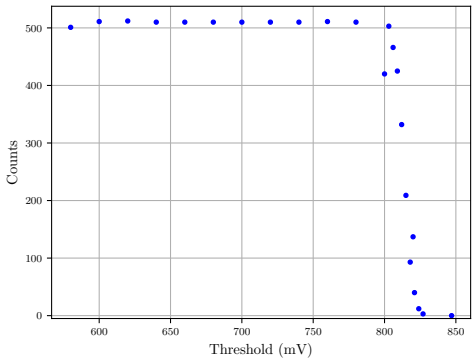

Figure A.11.: Threshold scan of the injection of 100 pixels

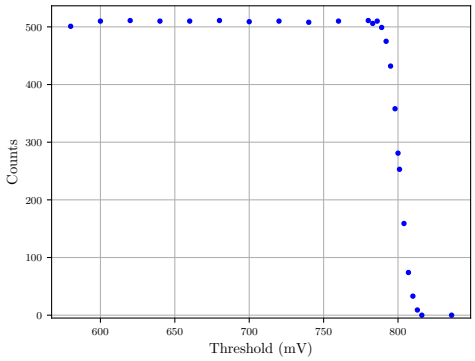

Figure A.12.: Threshold scan of the injection of 625 pixels

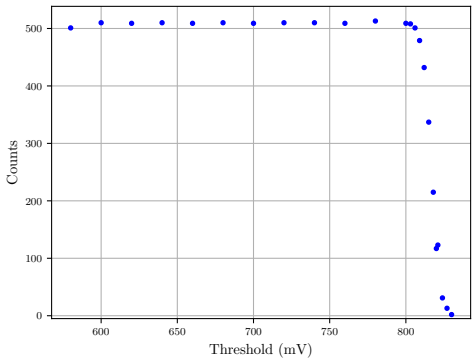

Figure A.13.: Threshold scan of the injection of 25 pixels

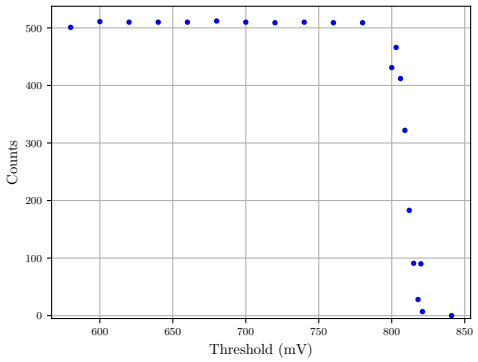

Figure A.14.: Threshold scan of the injection of 225 pixels

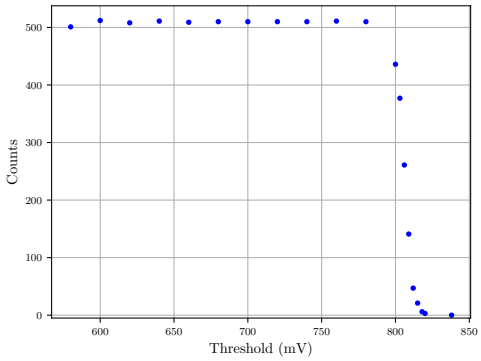

Figure A.15.: Threshold scan of the injection of 400 pixels

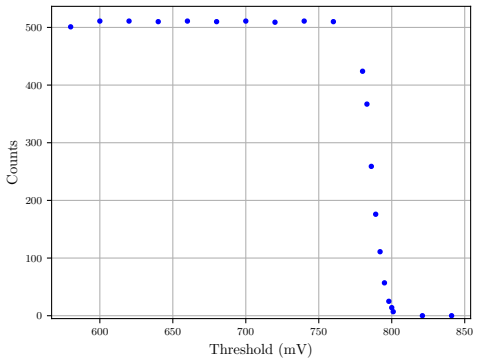

Figure A.16.: Threshold scan of the injection of 1225 pixels

A.3. Time delay measurements

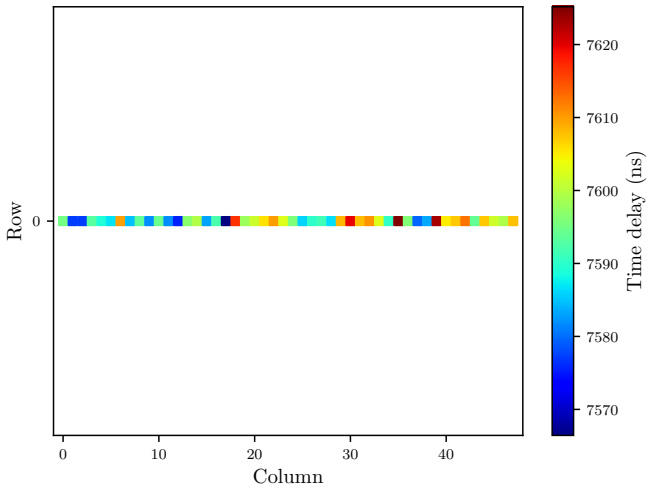

Figure A.17.: Time delay values for all pixels on row 0

A.4. Figures of the threshold scans for different values of TDAC1

These graphs correspond to the measurements discussed in section 3.6.1.

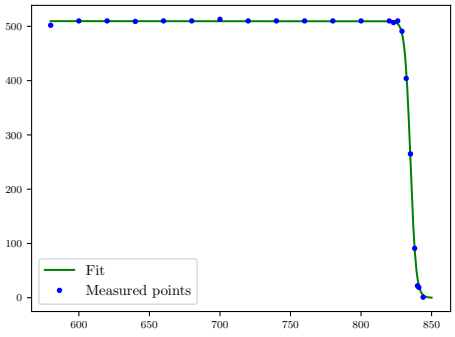

Figure A.18.: Measurement and fit of a threshold scan with TDAC1=0

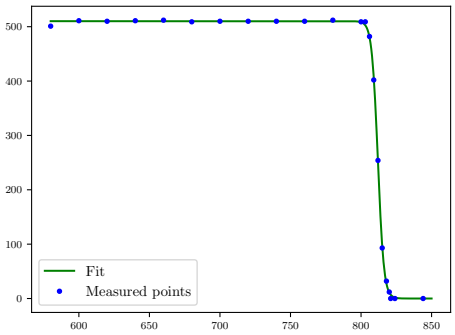

Figure A.19.: Measurement and fit of a threshold scan with TDAC1=4

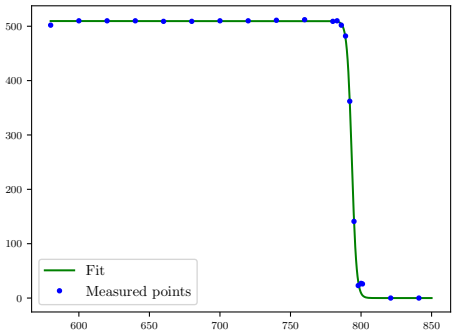

Figure A.20.: Measurement and fit of a threshold scan with TDAC1=7

A.5. Figures of the time delay measurement for different values of TDAC1

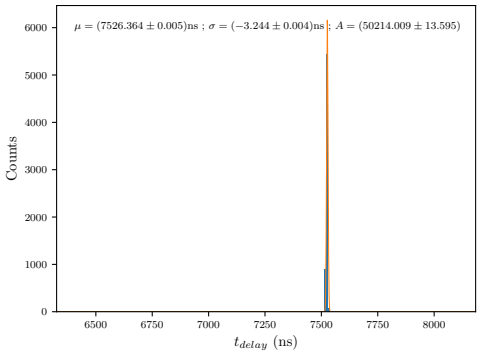

Figure A.21.: Time delay for TDAC1=0

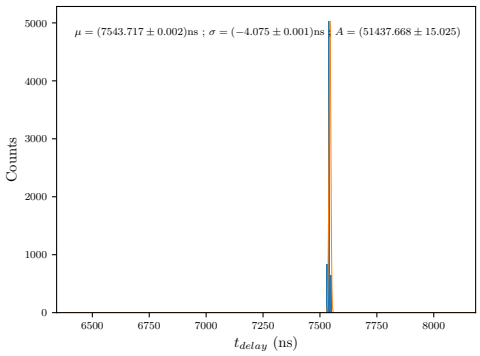

Figure A.22.: Time delay for TDAC1=4

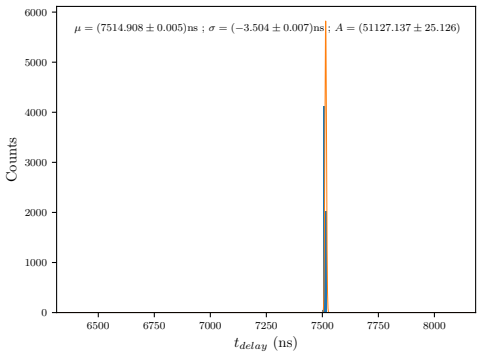

Figure A.23.: Time delay for TDAC1=7