Processing of triggerlessly acquired detector's data

PREPROCESSING

Load and prepare the dataset inside a Pandas' DataFrame.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from os import listdir
from os.path import isfile, join
from matplotlib.ticker import MultipleLocator
import seaborn as sns

%matplotlib inline
```

Inside each directory there are several files related to different test. Now we want to merge all of them into the same DataFrame.

In [2]:
```
# Load part of the dataset (all the dataset cause memory erro
r, sometimes also more than two part can cause it)
# List all files inside the directory
directory = "/data/Run000333/"
file_names = [file for file in listdir(directory) if isfile(j
oin(directory, file))]

# Create dataframe by appending the data from each file
data = pd.read_csv(directory + file_names[0])
# It's possible increase the range to load more data (up to l
en(file_names))
# In this case we will only use one part
for i in range(1, 1):
    data = data.append(pd.read_csv(directory + file_names[i
]))
data = data.reset_index(drop=True)
```

In [3]:
```
# Useful constants
Tmax = 390 # ns
L = 42 # mm
Vd = L/(2*Tmax) # mm/ns
pos_offset = 21 # mm

# Add column of time (ns)
# There is a problem with the precision of the measures, so w
e drop the orbit
# Real time: data['TIME_NS'] = data["ORBIT_CNT"]*3564*25 + da
ta["BX_COUNTER"]*25 + data["TDC_MEAS"]*25/30
data['TIME_NS'] = data["BX_COUNTER"]*25 + data["TDC_MEAS"]*25
/30
```

You signed in with another tab or window. Reload to refresh your session.

Out[3]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS	T
0	1	0	60	3879571901	681	0	1
1	1	0	59	3879571901	686	4	1
2	1	0	63	3879571901	819	8	2
3	1	0	64	3879571901	821	7	2
4	1	0	60	3879571901	907	25	2

To compute the constant t_0, which is different for every event, we can use the following relation:

$$T_{MAX} = \frac{t_1 + t_3}{2} + t_2$$

where $t_1 = t_{R_1} - t_0$, $t_2 = t_{R_2} - t_0$ and $t_3 = t_{R_3} - t_0$. Then the relation become:

$$T_{MAX} = \frac{t_{R_1} - t_0 + t_{R_3} - t_0}{2} + t_{R_2} - t_0$$

from which we get:

$$t_0 = \frac{t_{R_1} + t_{R_3} + 2t_{R_2} - 2T_{MAX}}{4}$$

Finally we notice that t_{R_1}, t_{R_2}, t_{R_3} are the times recorded by each cell, which are already available in our dataset.

Before processing the dataset, we have to create some missing columns, in fact the DataFrame with the events must contain the following information:

- CHAMBER, which is the Detector number [1-4];
- LAYER, which is the layer of the cell [1-4];
- CELL, which is in the number of the cell [1-16];
- POSTION, which is the position where a particle traverses the cell [0-21] (in mm).

Column of LAYER

To get the layer we can compute the remainder of the TDC_CHANNEL with 4 (total number of layers), and then we have to remap the values in the following way:

REMAINDER	LAYER
0	1
1	4
2	2
3	3

In [4]:
```
# To get the layer we must get the remainder of the TDC_CHANN
EL with 4
# Then we must reoder the result as described above
data['LAYER'] = data['TDC_CHANNEL'] % 4
```

```
# Map 0 -> 1
data.loc[data['LAYER'] == 0,'LAYER'] = 1

# Check the correctness
data.head(5)
```

Out[4]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS	T
0	1	0	60	3879571901	681	0	1
1	1	0	59	3879571901	686	4	1
2	1	0	63	3879571901	819	8	2
3	1	0	64	3879571901	821	7	2
4	1	0	60	3879571901	907	25	2

Column of CHAMBER

Create the column for the chamber according to the following rules:

- Detector 1 → FPGA 0, TDC_CHANNEL in [1-64]
- Detector 2 → FPGA 0, TDC_CHANNEL in [65-128]
- Detector 3 → FPGA 1, TDC_CHANNEL in [1-64]
- Detector 4 → FPGA 1, TDC_CHANNEL in [65-128]

In [5]:

```
# Create column for chamber
# Before create empty column
data['CHAMBER'] = 0

# Detector 1
# Select all rows with FPGA = 0 and TDC_CHANNEL <= 64
data.loc[(data['FPGA'] == 0) & (data['TDC_CHANNEL'] <= 64),'C
HAMBER'] = 1

# Detector 2
# Select all rows with FPGA = 0 and  64 < TDC_CHANNEL <= 128
data.loc[(data['FPGA'] == 0) & (data['TDC_CHANNEL'] > 64) & (
data['TDC_CHANNEL'] <= 128),'CHAMBER'] = 2

# Detector 3
# Select all rows with FPGA = 1 and TDC_CHANNEL <= 64
data.loc[(data['FPGA'] == 1) & (data['TDC_CHANNEL'] <= 64),
'CHAMBER'] = 3

# Detector 4
# Select all rows with FPGA = 0 and  64 < TDC_CHANNEL <= 128
data.loc[(data['FPGA'] == 1) & (data['TDC_CHANNEL'] > 64) & (
data['TDC_CHANNEL'] <= 128),'CHAMBER'] = 4

# Check the correctness
data.head(5)
```

Out[5]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS	T
0	1	0	60	3879571901	681	0	1

2	1	0	63	3879571901	819	8	2
3	1	0	64	3879571901	821	7	2
4	1	0	60	3879571901	907	25	2

Column of CELL

This column contains the values from 1 to 16. These values can be obtained as follows:

$$\lceil \frac{N_{CHANNEL}\%64}{4} \rceil$$

In [6]:

```
# Create column for chamber
data['CELL'] = ((data['TDC_CHANNEL']%64)/4).apply(np.ceil).as
type(int)
# TDC_CHANNEL%64=0 refers always to cell 16 of layer 1
data.loc[data['CELL']==0,'CELL'] = 16

# Check the correctness
data.head(5)
```

Out[6]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS	T
0	1	0	60	3879571901	681	0	1
1	1	0	59	3879571901	686	4	1
2	1	0	63	3879571901	819	8	2
3	1	0	64	3879571901	821	7	2
4	1	0	60	3879571901	907	25	2

PART 1

The dataset is ready to be processed, so we can start detecting the events through the trigger 139.

In [7]:

```
# Silence warning
pd.options.mode.chained_assignment = None  # default='warn'

# Search all the orbit with the trigger 139
orbit = data.loc[data['TDC_CHANNEL'] == 139,'ORBIT_CNT']
list_orbit = orbit.values.tolist()
events = data.loc[data['ORBIT_CNT'].isin(list_orbit)]

# Sort data
# It is wrong to order data according to TIME_NS because this
time depends on where the particel has passed
# inside the cell, in fact if for example the particel passes
ther cell of the third layer near its center
# the drift time will be small and so TIME_NS can result smal
ler than the one recorded by the cell in
# the first layer, where the particel has passed the cell far
away from its center.
# So we will sord values according to their OBIT_CNT, CHAMBER
```

```
# Remove unreal hit rows
events = events[events['TDC_CHANNEL']<129]

events.head(5)
```

Out[7]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS
0	1	0	60	3879571901	681	0
3	1	0	64	3879571901	821	7
4	1	0	60	3879571901	907	25
6	1	0	64	3879571901	905	22
12	1	0	56	3879571901	954	17

In [8]:

```
# Visualization hit matrix function
def show_mat(df):

    # Hit Matrix initialization
    raw_mat = np.zeros((16, 32))
    rows = np.array((df['CHAMBER']-1)*4+df['LAYER']-1)
    columns = np.array((df['CELL']-1)*2)
    for i in range(len(rows)):
        raw_mat[rows[i], columns[i]] = raw_mat[rows[i], colum
ns[i]]+1
        raw_mat[rows[i], columns[i]+1] = raw_mat[rows[i], col
umns[i]+1]+1

    # Reshape the hit matrix
    final_mat = np.zeros((8, 66))

    # Chamber 1,2
    for i in range(8):
        if (i%2 == 0):
            final_mat[7-i, 1:33] = raw_mat[i, :32]
        else:
            final_mat[7-i, :32] = raw_mat[i, :32]
    # Chamber 3,4
    for i in range(8, 16):
        if (i%2 == 0):
            final_mat[7-(i-8), 34:66] = raw_mat[i, :32]
        else:
            final_mat[7-(i-8), 33:65] = raw_mat[i, :32]


    # Showing the resutls
    plt.figure(figsize=(20,4))
    ax = plt.imshow(final_mat, cmap='plasma')

    plt.annotate('CHAMBER 1',xy=(0.5, 0.5), xytext=(12,11), f
ontsize=20, color='red')
    plt.annotate('CHAMBER 2',xy=(0.5, 0.5), xytext=(12,-3), f
ontsize=20, color='red')
    plt.annotate('CHAMBER 3',xy=(0.5, 0.5), xytext=(46,11), f
ontsize=20, color='red')
    plt.annotate('CHAMBER 4',xy=(0.5, 0.5), xytext=(46,-3), f
```

```
    plt.axhline(y=3.5, color='white', linewidth=2)

    plt.xticks(np.concatenate((np.arange(1.5, 33, 2 ), np.ara
nge(34.5, 66, 2))),
               ['1','2','3','4','5','6','7','8','9','10','11'
,'12','13','14','15','16',
                '1','2','3','4','5','6','7','8','9','10','11'
,'12','13','14','15','16'])
    plt.yticks([x for x in range(8)], ['4','3','2','1','4',
'3','2','1'])
    plt.ylabel('LAYER', fontsize=16)
    plt.xlabel('CELL', fontsize=16)
    plt.colorbar()

# Visualization single event
def show_event(df, eN):
    show_mat(df[df['EVENT_NUMBER']==eN])
```

Computation of t_0

To compute t_0 we have to apply the Talete's equation to the cell alignment inside our dataset. We will limit our search to the following patterns inside the 'LAYER' column:

- 1, 2, 3
- 2, 3, 4

This process can be easily generalized to other patterns.

In [9]:
```
# Make three shifted copy of the LAYER, TIME_NS, CELL and of
 CHAMBER
events['LAYER_1'] = events['LAYER'].shift(-1)
events['LAYER_2'] = events['LAYER'].shift(-2)
events['TIME_NS_1'] = events['TIME_NS'].shift(-1)
events['TIME_NS_2'] = events['TIME_NS'].shift(-2)
events['CELL_1'] = events['CELL'].shift(-1)
events['CELL_2'] = events['CELL'].shift(-2)
events['CHAMBER_1'] = events['CHAMBER'].shift(-1)
events['CHAMBER_2'] = events['CHAMBER'].shift(-2)

# There is a constrains on Chambers, which must be always the
same
mask_pattern_chamber = (events['CHAMBER'] == events['CHAMBER_
1']) & (events['CHAMBER'] == events['CHAMBER_2'])

# There is also a constrains on cell, which must be applied f
or every pattern
# The sucessive cell hit number has to differ at most by one
 from the previous cell number (1->2 ok, 6->5 ok, 1->3 no)
mask_pattern_cell_123 = ((events['CELL_1']-events['CELL'])>=0
& (abs(events['CELL_1']-events['CELL'])<=1)) & (((events['CEL
L_2']-events['CELL_1'])<=0) & (abs(events['CELL_2']-events['C
ELL_1'])<=1))
mask_pattern_cell_234 = ((events['CELL_1']-events['CELL'])<=0
& (abs(events['CELL_1']-events['CELL'])<=1)) & (((events['CEL
L_2']-events['CELL_1'])>=0) & (abs(events['CELL_2']-events['C
ELL_1'])<=1))
```

```
=2) & (events['LAYER_2']==3)
mask_pattern_234 = (events['LAYER']==2) & (events['LAYER_1']=
=3) & (events['LAYER_2']==4)

# Compute final mask
mask_pattern = ((mask_pattern_123 & mask_pattern_cell_123) |
(mask_pattern_234 & mask_pattern_cell_234)) & mask_pattern_ch
amber

# Apply the Talete's Theorem for t0 to the patterns
events.loc[mask_pattern, 't0'] = (events['TIME_NS'] + events[
'TIME_NS_2'] + 2*events['TIME_NS_1'] - 2*Tmax)/4

# Populate values of adiacent cell (according to the choosen
 pattern)
events = events.fillna(0)
mask_pattern_time = events['t0']!=0
events.loc[mask_pattern_time.shift(1).fillna(False), 't0'] =
events['t0'].shift(1)
events.loc[mask_pattern_time.shift(2).fillna(False), 't0'] =
events['t0'].shift(2)
events = events.fillna(0)

events.head(5)
```

Out[9]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS
0	1	0	60	3879571901	681	0
3	1	0	64	3879571901	821	7
4	1	0	60	3879571901	907	25
6	1	0	64	3879571901	905	22
12	1	0	56	3879571901	954	17

Column POSITION

Only at this point we can create the column with the position, thanks to t_0.

In [10]:

```
# Compute the position
events.loc[events['t0']!=0,'POSITION'] = (events['TIME_NS'] -
events['t0'])*Vd
events = events.fillna(0)

# Now we have to filter false alignmens detected by the mask
 pattern, which can be caused by noise
# So we drop the rows with an unexpected result: such as Posi
tion bigger than 21 mm or smaller than 0 mm
events.loc[(events['POSITION']<0) | (events['POSITION']>=21),
['POSITION','t0']] = 0

events.head(5)
```

Out[10]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS

4	1	0	60	3879571901	907	25
6	1	0	64	3879571901	905	22
12	1	0	56	3879571901	954	17

In [11]:

```
# Map obit values to a range of int
grouped_orbit = events.groupby('ORBIT_CNT')
# Search all orbits
orbits = list(grouped_orbit.groups.keys())
# Create increasing number list for the events
event_number = np.arange(1, len(orbits)+1)
# Create the map
event_map = dict(zip(orbits, event_number))
# Map values
orbit_to_map = events['ORBIT_CNT']
orbit_mapped = orbit_to_map.map(event_map)
events['EVENT_NUMBER'] = orbit_mapped

events.head(5)
```

Out[11]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEAS
0	1	0	60	3879571901	681	0
3	1	0	64	3879571901	821	7
4	1	0	60	3879571901	907	25
6	1	0	64	3879571901	905	22
12	1	0	56	3879571901	954	17

5 rows × 21 columns

In [12]:

```
# Final DataFrame showing all the events detected by the 139
 trigger
# When the position is not avaiable is set to 0
events_final = events[['EVENT_NUMBER','ORBIT_CNT','CHAMBER',
'LAYER','CELL','POSITION']]
events_final.set_index(['EVENT_NUMBER','ORBIT_CNT','CHAMBER',
'LAYER'], inplace=True)
events_final.sort_index(inplace=True)
events_final.head(10)
```

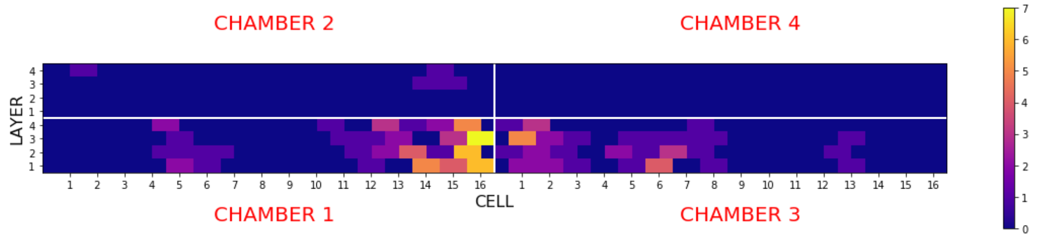
Out[12]:

				CELL	POSITION
EVENT_NUMBER	**ORBIT_CNT**	**CHAMBER**	**LAYER**		
			1	15	0.0
			1	16	0.0
			1	15	0.0
			1	16	0.0
1	**3879571901**	**1**	**1**	14	0.0

			1	16	0.0
			1	14	0.0
			1	16	0.0

Show the cell alingment inside an event

In [13]:
```
show_event(events, 1)
```



Plot the distribution of Drift Times

To compute the drift time we easily do:

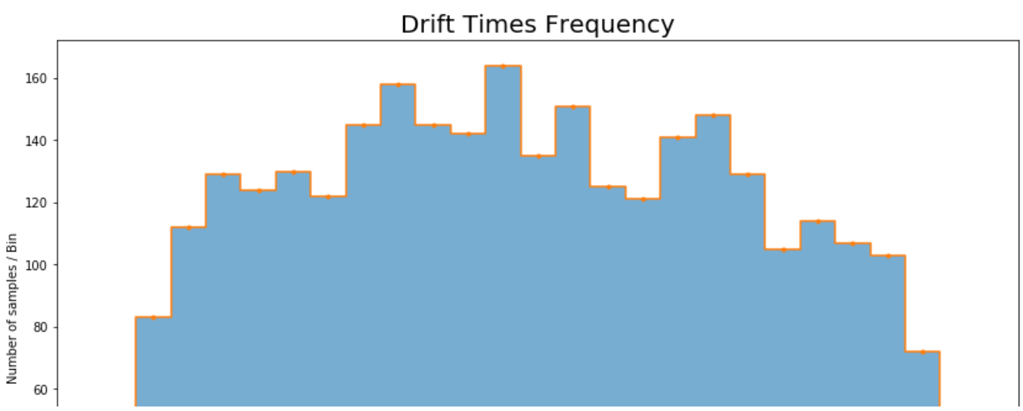
$$t = t_R - t_0$$

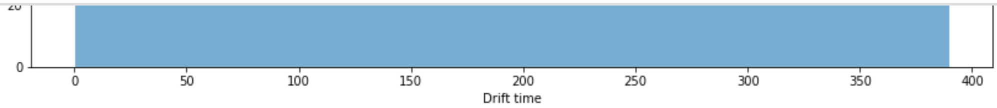
where t_R is the recorded time by each cell.

In [14]:
```
# Select hits where position is different from 0
events_drift = events[events['POSITION']!=0]
# Get all drift times
drift_times = events_drift['TIME_NS']-events_drift['t0']

# Drift Times Frequency
figure = plt.figure(figsize=(14,8))
ax = figure.add_subplot(111)
number_bins = 25
y, edges, bins = ax.hist(drift_times, bins = number_bins, lab
el='Drift Times Frequency', alpha=0.6)
ax.set_ylabel("Number of samples / Bin")
ax.set_xlabel("Drift time")
ax.set_title("Drift Times Frequency", fontsize=20)
mean_point = (edges[1:] + edges[:-1])/2
ax.errorbar(mean_point, y, yerr = y**-0.5, marker = '.', draw
style = 'steps-mid', label = 'error')

plt.show()
```





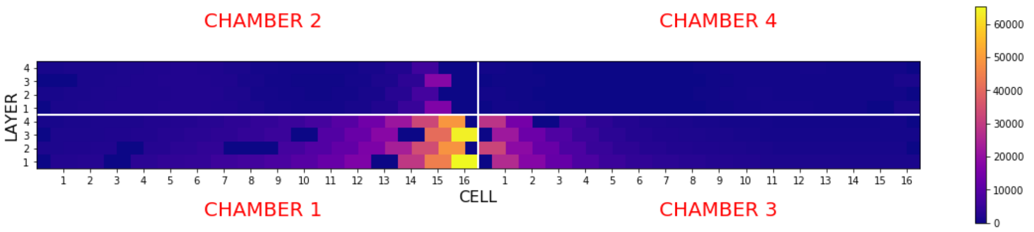
Plot of the Dataframe

Graphical visalization of all the trajectory of the particels.

In [15]:
```
# Visualize all the hit from orbits marked by trigger 139
print('Number of total hit detected:', data[data['TDC_CHANNE
L']<129].shape[0])
print('Number of hit belonging to orbit marked by trigger 13
9:', events.shape[0]/data[data['TDC_CHANNEL']<129].shape[0]*1
00, '%')
print('Number of events detected using the trigger:', len(eve
nt_number))

show_mat(events)
```

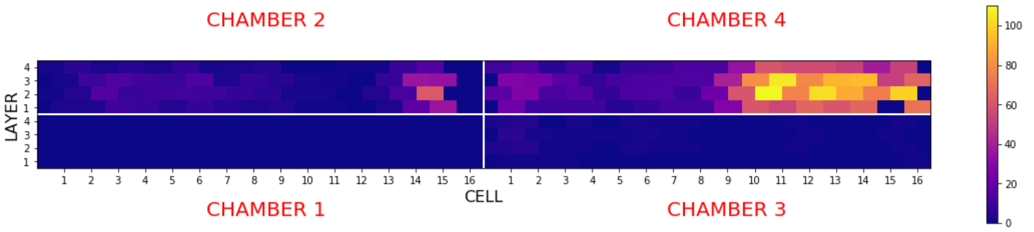
```
Number of total hit detected: 1202490
Number of hit belonging to orbit marked by trigger 139: 99.99
750517675822 %
Number of events detected using the trigger: 9480
```



In [16]:
```
# Visualize only hit with a legit calculated position
print('Number of hit which show a legit calculated position:'
, events[events['POSITION']!=0].shape[0]/data[data['TDC_CHANN
EL']<129].shape[0]*100, '%')

show_mat(events[events['POSITION']!=0])
```

```
Number of hit which show a legit calculated position: 0.24914
968107842891 %
```



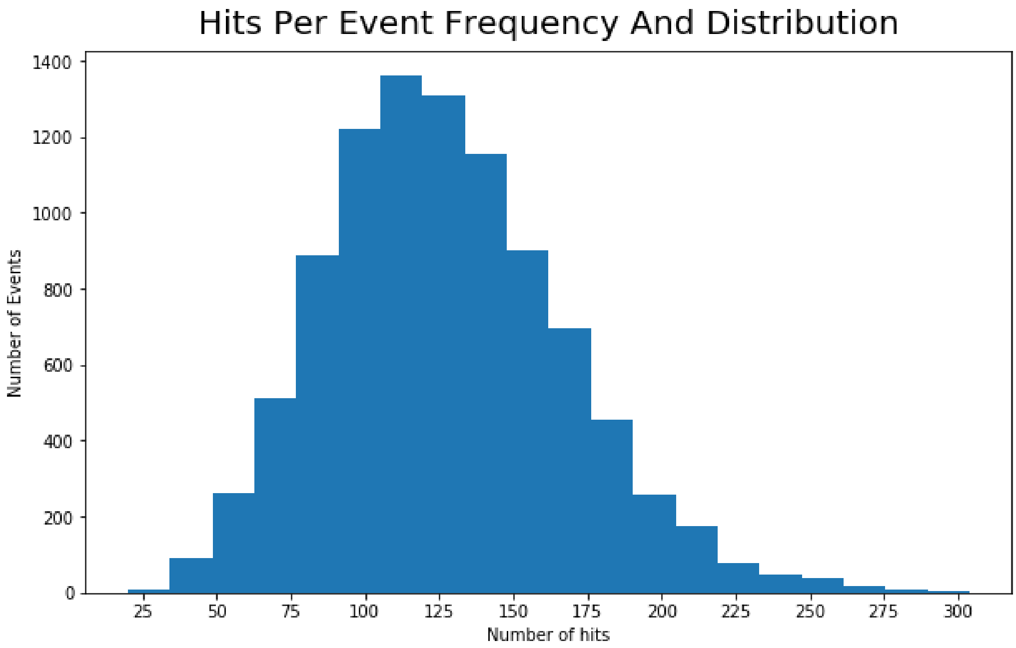
Plot of the Hits per Event

Histogram of the number of hit per event, considering all the hits belonging to triggered orbit

In [17]:
```
# Group the events by event number
grouped_ev_num = events.groupby('EVENT_NUMBER')
```

```
# For every event count the number of hits (i.e. return the l
enght of the array containing the hits of a certain event)
for i in grouped_ev_num.groups.keys():
    hits_per_event.append(len(np.array(grouped_ev_num.groups[
i])))
hits_per_event = np.array(hits_per_event)
```

In [18]:
```
# Plot the distribution of Hits/Event
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(1, 1, 1)
ax.xaxis.set_major_locator(MultipleLocator(25.000))
ax.set_title("Hits Per Event Frequency And Distribution", fon
tsize=20, verticalalignment='bottom')
ax.set_xlabel('Number of hits')
ax.set_ylabel('Number of Events')
ax.hist(hits_per_event, bins=20);
#sns.distplot(hits_per_event, bins=25, ax=ax);
```



RETRY THE SAME PROCEDURE USING TRIGGER 137

In [19]:
```
# Silence warning
pd.options.mode.chained_assignment = None  # default='warn'

# Search all the orbit with the trigger 137
orbit = data.loc[data['TDC_CHANNEL'] == 137,'ORBIT_CNT']
list_orbit = orbit.values.tolist()
events = data.loc[data['ORBIT_CNT'].isin(list_orbit)]

# Sort data
# It is wrong to order data according to TIME_NS because this
time depends on where the particel has passed
# inside the cell, in fact if for example the particel passes
ther cell of the third layer near its center
# the drift time will be small and so TIME_NS can result smal
ler than the one recorded by the cell in
```

```
and LAYER
events = events.sort_values(by = ['ORBIT_CNT','CHAMBER','LAYE
R'])

# Remove unreal hit rows
events = events[events['TDC_CHANNEL']<129]

events.head(5)
```

Out[19]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEA
32968	1	0	48	3879572127	17	26
32975	1	0	56	3879572127	681	1
32977	1	0	48	3879572127	688	19
32989	1	0	32	3879572127	1407	1
32995	1	0	60	3879572127	1434	19

Computation of t_0

To compute t_0 we have to apply the Talete's equation to the cell alignment inside our dataset. We will limit our search to the following patterns inside the 'LAYER' column:

- 1, 2, 3
- 2, 3, 4

This process can be easily generalized to other patterns.

In [20]:

```
# Make three shifted copy of the LAYER, TIME_NS, CELL and of
 CHAMBER
events['LAYER_1'] = events['LAYER'].shift(-1)
events['LAYER_2'] = events['LAYER'].shift(-2)
events['TIME_NS_1'] = events['TIME_NS'].shift(-1)
events['TIME_NS_2'] = events['TIME_NS'].shift(-2)
events['CELL_1'] = events['CELL'].shift(-1)
events['CELL_2'] = events['CELL'].shift(-2)
events['CHAMBER_1'] = events['CHAMBER'].shift(-1)
events['CHAMBER_2'] = events['CHAMBER'].shift(-2)

# There is a constrains on Chambers, which must be always the
same
mask_pattern_chamber = (events['CHAMBER'] == events['CHAMBER_
1']) & (events['CHAMBER'] == events['CHAMBER_2'])

# There is also a constrains on cell, which must be applied f
or every pattern
# The sucessive cell hit number has to differ at most by one
 from the previous cell number (1->2 ok, 6->5 ok, 1->3 no)
mask_pattern_cell_123 = ((events['CELL_1']-events['CELL'])>=0
& (abs(events['CELL_1']-events['CELL'])<=1)) & (((events['CEL
L_2']-events['CELL_1'])<=0) & (abs(events['CELL_2']-events['C
ELL_1'])<=1))
mask_pattern_cell_234 = ((events['CELL_1']-events['CELL'])<=0
```

⚠ You signed in with another tab or window. Reload to refresh your session.

```
# Search pattern to get the real t0
mask_pattern_123 = (events['LAYER']==1) & (events['LAYER_1']==2) & (events['LAYER_2']==3)
mask_pattern_234 = (events['LAYER']==2) & (events['LAYER_1']==3) & (events['LAYER_2']==4)

# Compute final mask
mask_pattern = ((mask_pattern_123 & mask_pattern_cell_123) | (mask_pattern_234 & mask_pattern_cell_234)) & mask_pattern_chamber

# Apply the Talete's Theorem for t0 to the patterns
events.loc[mask_pattern, 't0'] = (events['TIME_NS'] + events['TIME_NS_2'] + 2*events['TIME_NS_1'] - 2*Tmax)/4

# Populate values of adiacent cell (according to the choosen pattern)
events = events.fillna(0)
mask_pattern_time = events['t0']!=0
events.loc[mask_pattern_time.shift(1).fillna(False), 't0'] = events['t0'].shift(1)
events.loc[mask_pattern_time.shift(2).fillna(False), 't0'] = events['t0'].shift(2)
events = events.fillna(0)

events.head(5)
```

Out[20]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEA
32968	1	0	48	3879572127	17	26
32975	1	0	56	3879572127	681	1
32977	1	0	48	3879572127	688	19
32989	1	0	32	3879572127	1407	1
32995	1	0	60	3879572127	1434	19

Column POSITION

Only at this point we can create the column with the position, thanks to t_0.

In [21]:
```
# Compute the position
events.loc[events['t0']!=0,'POSITION'] = (events['TIME_NS'] - events['t0'])*Vd
events = events.fillna(0)

# Now we have to filter false alignmens detected by the mask pattern, which can be caused by noise
# So we drop the rows with an unexpected result: such as Position bigger than 21 mm or smaller than 0 mm
events.loc[(events['POSITION']<0) | (events['POSITION']>=21), ['POSITION','t0']] = 0

events.head(5)
```

32968	1	0	48	3879572127	17	26
32975	1	0	56	3879572127	681	1
32977	1	0	48	3879572127	688	19
32989	1	0	32	3879572127	1407	1
32995	1	0	60	3879572127	1434	19

In [22]:

```
# Map obit values to a range of int
grouped_orbit = events.groupby('ORBIT_CNT')
# Search all orbits
orbits = list(grouped_orbit.groups.keys())
# Create increasing number list for the events
event_number = np.arange(1, len(orbits)+1)
# Create the map
event_map = dict(zip(orbits, event_number))
# Map values
orbit_to_map = events['ORBIT_CNT']
orbit_mapped = orbit_to_map.map(event_map)
events['EVENT_NUMBER'] = orbit_mapped

events.head(5)
```

Out[22]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEA
32968	1	0	48	3879572127	17	26
32975	1	0	56	3879572127	681	1
32977	1	0	48	3879572127	688	19
32989	1	0	32	3879572127	1407	1
32995	1	0	60	3879572127	1434	19

5 rows × 21 columns

In [23]:

```
# Final DataFrame showing all the events detected by the 137
 trigger
# When the position is not avaiable is set to 0
events_final = events[['EVENT_NUMBER','ORBIT_CNT','CHAMBER',
'LAYER','CELL','POSITION']]
events_final.set_index(['EVENT_NUMBER','ORBIT_CNT','CHAMBER',
'LAYER'], inplace=True)
events_final.sort_index(inplace=True)
events_final.head(10)
```

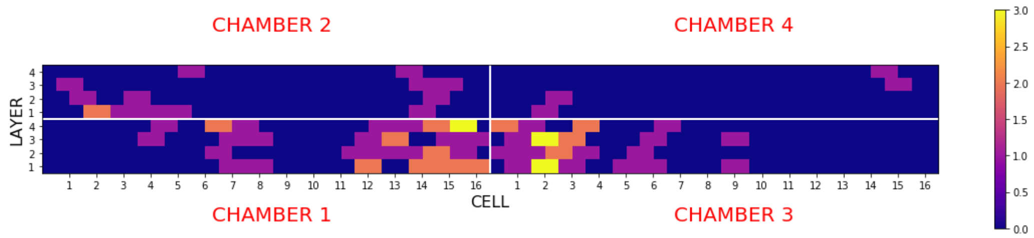
Out[23]:

				CELL	POSITION
EVENT_NUMBER	**ORBIT_CNT**	**CHAMBER**	**LAYER**		
			1	12	0.0
			1	14	0.0
			1	12	0.0
			1	8	0.0

			1	16	0.0
			1	7	0.0
			1	15	0.0
			1	14	0.0

Show the cell alingment inside an event

In [24]:
```
show_event(events, 1)
```



Plot the distribution of Drift Times

To compute the drift time we easily do:

$$t = t_R - t_0$$

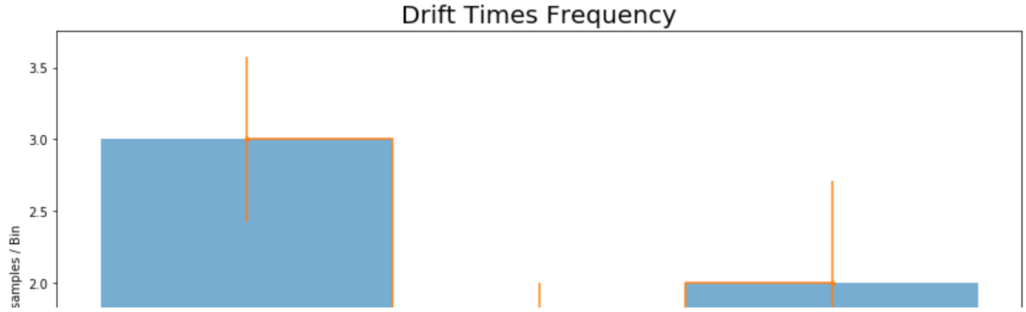
where t_R is the recorded time by each cell.

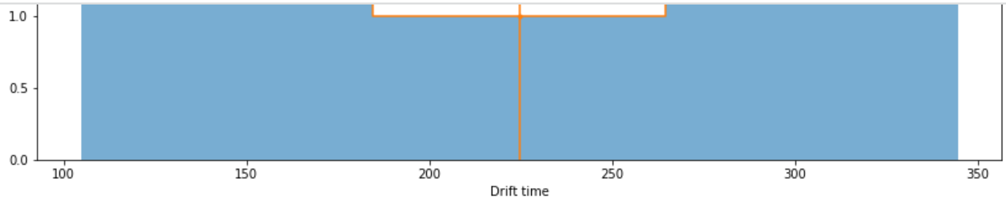
In [25]:
```
# Select hits where position is different from 0
events_drift = events[events['POSITION']!=0]
# Get all drift times
drift_times = events_drift['TIME_NS']-events_drift['t0']

# Drift Times Frequency
figure = plt.figure(figsize=(14,8))
ax = figure.add_subplot(111)
number_bins = 3
y, edges, bins = ax.hist(drift_times, bins = number_bins, lab
el='Drift Times Frequency', alpha=0.6)
ax.set_ylabel("Number of samples / Bin")
ax.set_xlabel("Drift time")
ax.set_title("Drift Times Frequency", fontsize=20)
mean_point = (edges[1:] + edges[:-1])/2
ax.errorbar(mean_point, y, yerr = y**-0.5, marker = '.', draw
style = 'steps-mid', label = 'error')

plt.show()
```

Drift Times Frequency





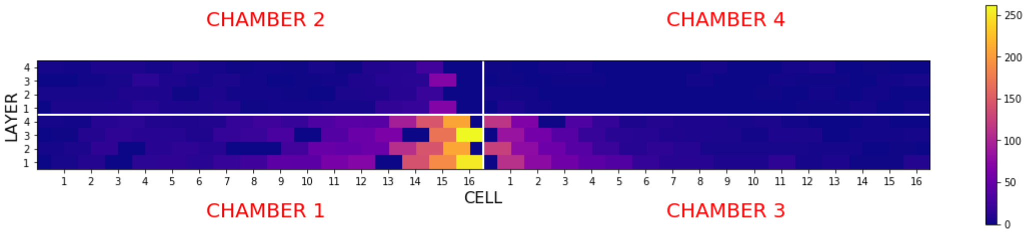
Plot of the Dataframe

Graphical visalization of all the trajectory of the particels.

In [26]:
```
# Visualize all the hit from orbits marked by trigger 137
print('Number of total hit detected:', data[data['TDC_CHANNEL']<129].shape[0])
print('Number of hit belonging to orbit marked by trigger 137:', events.shape[0]/data[data['TDC_CHANNEL']<129].shape[0]*100, '%')
print('Number of events detected using the trigger:', len(event_number))

show_mat(events)
```

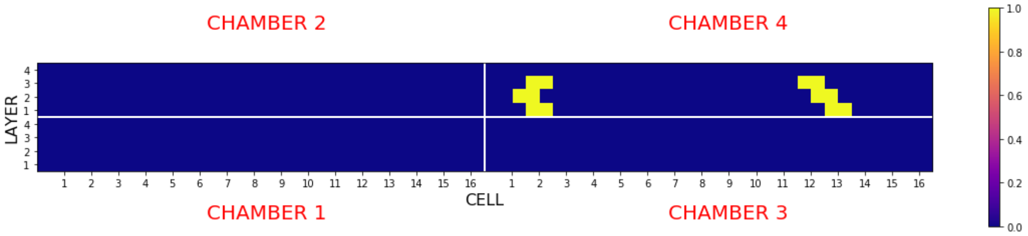
```
Number of total hit detected: 1202490
Number of hit belonging to orbit marked by trigger 137: 0.41522174820580626 %
Number of events detected using the trigger: 37
```



In [27]:
```
# Visualize only hit with a legit calculated position
print('Number of hit which show a legit calculated position:', events[events['POSITION']!=0].shape[0])

show_mat(events[events['POSITION']!=0])
```

```
Number of hit which show a legit calculated position: 6
```



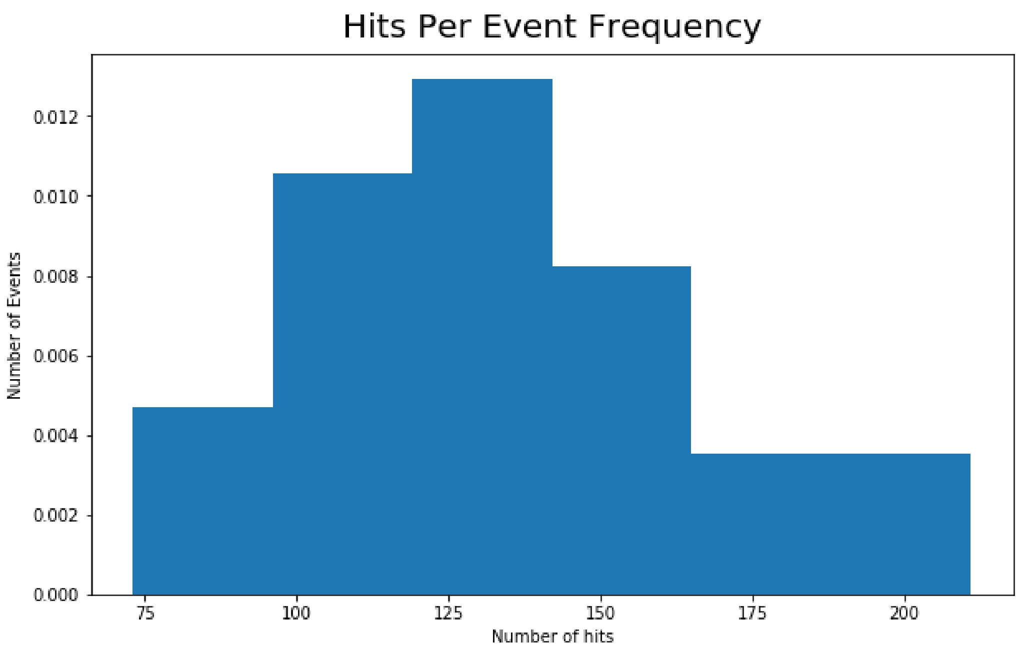
Plot of the Hits per Event

Histogram of the number of hit per event, considering all the hits belonging to triggered orbit

In [28]:
```
# Group the events by event number
grouped_ev_num = events.groupby('EVENT_NUMBER')
```

```
# For every event count the number of hits (i.e. return the l
enght of the array containing the hits of a certain event)
for i in grouped_ev_num.groups.keys():
    hits_per_event.append(len(np.array(grouped_ev_num.groups[
i])))
hits_per_event = np.array(hits_per_event)
```

In [29]:
```
# Plot the distribution of Hits/Event
fig = plt.figure(figsize=(10,6))
ax = fig.add_subplot(1, 1, 1)
ax.xaxis.set_major_locator(MultipleLocator(25.000))
ax.set_title("Hits Per Event Frequency", fontsize=20, vertica
lalignment='bottom')
ax.set_xlabel('Number of hits')
ax.set_ylabel('Number of Events')
#sns.distplot(hits_per_event, bins=25, ax=ax);
ax.hist(hits_per_event, bins=6, density=True);
```



PART 2

Now we have to repeat the previous computation without the help of an external trigger, so we restart from the dataframe 'data'. To find the events we can use the orbits, the previous pattern and the mean trigger equation.

In [30]:
```
# Remove unreal hit rows and sort the dataframe
data = data[data['TDC_CHANNEL']<129]
data_length = data.shape[0]
data = data.sort_values(by = ['ORBIT_CNT','CHAMBER','LAYER'])

# Make three shifted copy of the LAYER, TIME_NS, CELL and of
 CHAMBER
data['LAYER_1'] = data['LAYER'].shift(-1)
data['LAYER_2'] = data['LAYER'].shift(-2)
data['TIME_NS_1'] = data['TIME_NS'].shift(-1)
data['TIME_NS_2'] = data['TIME_NS'].shift(-2)
```

```
data['CHAMBER_2'] = data['CHAMBER'].shift(-2)

# There is a constrains on Chambers, which must be always the
same
mask_pattern_chamber = (data['CHAMBER'] == data['CHAMBER_1'])
& (data['CHAMBER'] == data['CHAMBER_2'])

# There is also a constrains on cell, which must be applayed
 for every pattern
mask_pattern_cell = (abs(data['CELL_1']-data['CELL'])<2) & (a
bs(data['CELL_2']-data['CELL_1'])<2)

# Search pattern to get the real t0
mask_pattern_123 = (data['LAYER']==1) & (data['LAYER_1']==2)
& (data['LAYER_2']==3)
mask_pattern_234 = (data['LAYER']==2) & (data['LAYER_1']==3)
& (data['LAYER_2']==4)

# Compute final mask
mask_pattern = (mask_pattern_123 | mask_pattern_234) & mask_p
attern_cell & mask_pattern_chamber

# Apply the Talete's Theorem for t0 to the patterns
data.loc[mask_pattern, 't0'] = (data['TIME_NS'] + data['TIME_
NS_2'] + 2*data['TIME_NS_1'] - 2*Tmax)/4

# Populate values of adiacent cell (according to the choosen
 pattern)
data = data.fillna(0)
mask_pattern_time = data['t0']!=0
data.loc[mask_pattern_time.shift(1).fillna(False), 't0'] = da
ta['t0'].shift(1)
data.loc[mask_pattern_time.shift(2).fillna(False), 't0'] = da
ta['t0'].shift(2)
data = data.fillna(0)


# Compute the position
data.loc[data['t0']!=0,'POSITION'] = (data['TIME_NS'] - data[
't0'])*Vd
data = data.fillna(0)

# Now we have to filter false event, which can be caused by n
oise
# So we drop the rows with an unexpected result: such as Posi
tion bigger than 21 mm or smaller than 0 mm
data.loc[(data['POSITION']<0) | (data['POSITION']>=21), ['POS
ITION','t0']] = 0

data = data[data['POSITION']!=0]

data.head(5)
```

Out[30]:

	HEAD	FPGA	TDC_CHANNEL	ORBIT_CNT	BX_COUNTER	TDC_MEA!
565	1	1	106	3879571903	3157	26
569	1	1	107	3879571903	3166	25

1835	1	1	106	3879571911	656	22

In [31]:

```
# Recreate final events dataframe
# Map obit values to a range of int
grouped_orbit = data.groupby('ORBIT_CNT')
# Search all orbits
orbits = list(grouped_orbit.groups.keys())
# Create increasing number list for the events
event_number = np.arange(1, len(orbits)+1)
# Create the map
event_map = dict(zip(orbits, event_number))
# Map values
orbit_to_map = data['ORBIT_CNT']
orbit_mapped = orbit_to_map.map(event_map)
data['EVENT_NUMBER'] = orbit_mapped

events_final = data[['EVENT_NUMBER','ORBIT_CNT','CHAMBER','LA
YER','CELL','POSITION']]
events_final.set_index(['EVENT_NUMBER','ORBIT_CNT','CHAMBER',
'LAYER'], inplace=True)
events_final.sort_index(inplace=True)
events_final.head(10)
```

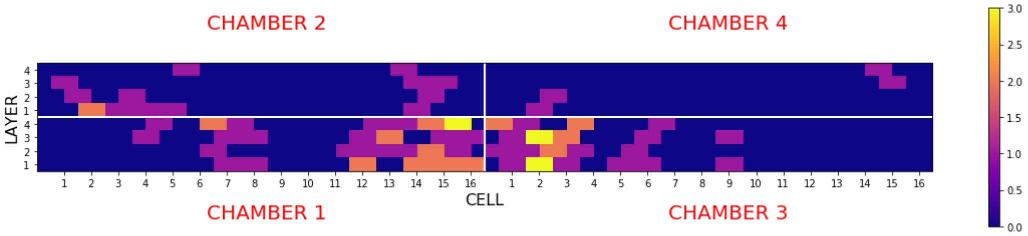
Out[31]:

				CELL	**POSITION**
EVENT_NUMBER	**ORBIT_CNT**	**CHAMBER**	**LAYER**		
1	**3879571903**	**4**	**2**	11	2.883013
			3	11	14.953526
			4	12	9.209936
2	**3879571911**	**4**	**1**	11	17.746795
			2	11	9.400641
			3	11	5.451923
3	**3879571912**	**4**	**2**	1	2.849359
			3	1	14.516026
			4	1	10.118590
4	**3879571922**	**4**	**2**	10	15.458333

Show the cell alingment inside an event

In [32]:

```
show_event(events, 1)
```

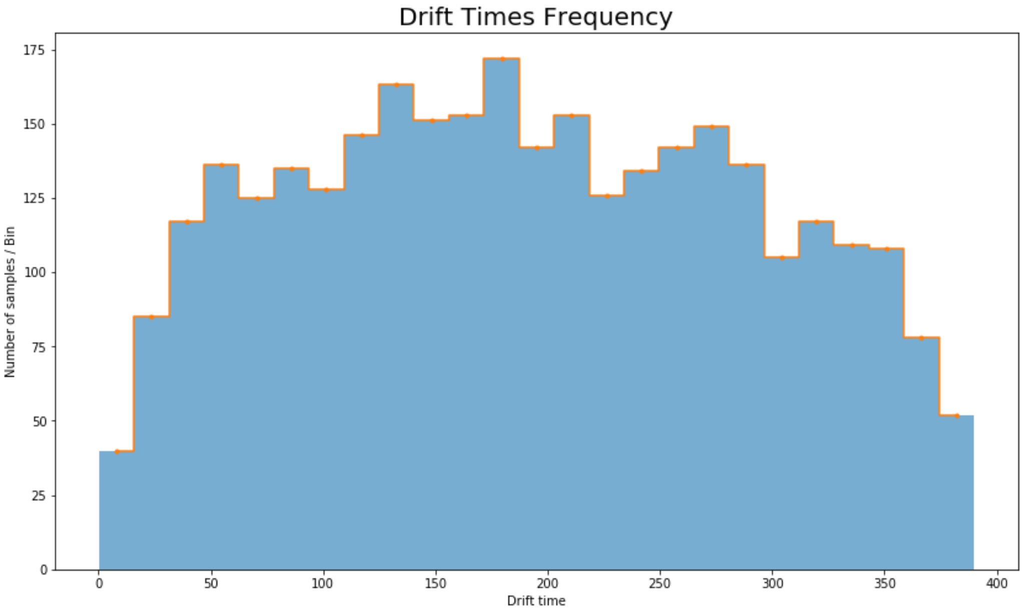


Plot the distribution of Drift Times

In [33]:
```
# Select hits where position is different from 0
events_drift = data
# Get all drift times
drift_times = events_drift['TIME_NS']-events_drift['t0']

# Drift Times Frequency
figure = plt.figure(figsize=(14,8))
ax = figure.add_subplot(111)
number_bins = 25
y, edges, bins = ax.hist(drift_times, bins = number_bins, lab
el='Drift Times Frequency', alpha=0.6)
ax.set_ylabel("Number of samples / Bin")
ax.set_xlabel("Drift time")
ax.set_title("Drift Times Frequency", fontsize=20)
mean_point = (edges[1:] + edges[:-1])/2
ax.errorbar(mean_point, y, yerr = y**-0.5, marker = '.', draw
style = 'steps-mid', label = 'error')

plt.show()
```

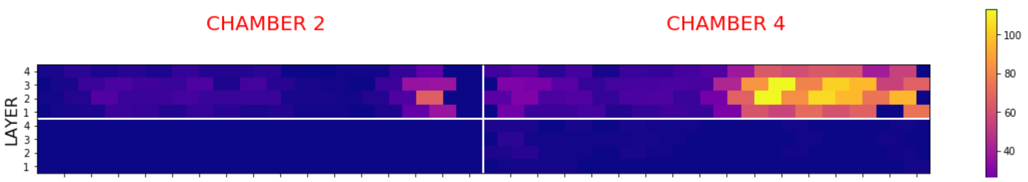


Plot of the Dataframe

In [34]:
```
print('Number of hit belongig to an event:', data.shape[0]/da
ta_length*100, '%')
print('Number of events detected using the trigger:', len(eve
nt_number))

show_mat(data)
```

```
Number of hit belongig to an event: 0.2579647231993613 %
Number of events detected using the trigger: 972
```



Plot of the Hits per Event

In [35]:
```
# Group the events by event number
grouped_ev_num = data.groupby('EVENT_NUMBER')

# Create a new list containing the number of hits per event
hits_per_event = []

# For every event count the number of hits (i.e. return the l
```