In [1]:
```
import sys
import pandas as pd
import time as time
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from matplotlib.lines import Line2D
import numba
import scienceplots
plt.style.use('science')
```

Load data

In [2]:
```
# Main detectors
dssd = pd.read_csv('processed_data/long_run_4mbar_500V/r47/dssd_non_vetoed_events
ppac = pd.read_csv('processed_data/long_run_4mbar_500V/r47/ppac_events.csv') # ra
ruth = pd.read_csv('processed_data/long_run_4mbar_500V/r47/rutherford_events.csv'

# DSSD regions
imp = dssd[dssd['event_type'] == 'imp']
boxE = dssd[dssd['event_type'] == 'boxE']
boxW = dssd[dssd['event_type'] == 'boxW']
boxT = dssd[dssd['event_type'] == 'boxT']
boxB = dssd[dssd['event_type'] == 'boxB']

# PPAC
cathode = ppac[ppac['detector'] == 'cathode']
anodeV = ppac[ppac['detector'] == 'anodeV']
anodeH = ppac[ppac['detector'] == 'anodeH']

# Rutherfords
ruth_E = ruth[ruth['detector'] == 'ruthE']
ruth_W = ruth[ruth['detector'] == 'ruthW']
```

PPAC-SHREC coincidences

In [3]:
```
# Coincidence window
window_before_ns = 5000  # 1700 ns (1.7 us) before
window_after_ns = 0   # 1000 ns (1 us) after

# Convert to picoseconds for use with timetag values
window_before_ps = window_before_ns * 1000  # ns to ps
window_after_ps = window_after_ns * 1000    # ns to ps
```

In [4]:
```
# Sort dfs by time (should already be sorted)
cathode_sorted = cathode.sort_values('timetag').reset_index(drop=True)
anodeV_sorted = anodeV.sort_values('timetag').reset_index(drop=True)
anodeH_sorted = anodeH.sort_values('timetag').reset_index(drop=True)
imp_sorted = imp.sort_values('tagx').reset_index(drop=True)  # Using tagx for IMP
```

In [5]:
```
# Grab timetag vals (faster searching)
cathode_timetags = cathode_sorted['timetag'].values
anodeV_timetags = anodeV_sorted['timetag'].values
anodeH_timetags = anodeH_sorted['timetag'].values
imp_timetags = imp_sorted['tagx'].values  # Using tagx as the IMP timetag
```

In [6]:
```
# Function to find PPAC events within the time window
def find_events_in_window(imp_timetag, detector_timetags, window_before_ps, windo
    """
    Find ppac events that occur within the specified time window around the IMP e
    All time values are in picoseconds.

    Params:
    -----------
    imp_timetag : Timestamp of the IMP event in picoseconds
    detector_timetags : Array of detector timestamps in picoseconds
    window_before_ps : Time window before the IMP event in picoseconds
    window_after_ps : Time window after the IMP event in picoseconds

    Returns:
    --------
    Indices of events within the window
    """
    # Calculate the time bounds
    lower_bound = imp_timetag - window_before_ps  # Time window before IMP
    upper_bound = imp_timetag + window_after_ps   # Time window after IMP

    # Find all events within these bounds using binary search
    lower_idx = np.searchsorted(detector_timetags, lower_bound)
    upper_idx = np.searchsorted(detector_timetags, upper_bound)

    if upper_idx > lower_idx:
        return list(range(lower_idx, upper_idx))
    return []

# Start timing the search
start_time = time.time()

# Create list to store coincident events
coincident_events = []
non_ppac_coincident_events = []

# Number of IMP events to process
total_imp_events = len(imp_sorted)
print(f"Processing {total_imp_events} IMP events...")

# For each IMP event, find coincident PPAC signals
for idx, imp_row in imp_sorted.iterrows():
    imp_timetag = imp_row['tagx'] # remember we are using tagx for the timetag he

    # Find ppac events in time window
    cathode_indices = find_events_in_window(imp_timetag, cathode_timetags, window
    anodeV_indices = find_events_in_window(imp_timetag, anodeV_timetags, window_b
    anodeH_indices = find_events_in_window(imp_timetag, anodeH_timetags, window_b

    # Only proceed if we have coincidences in all three PPAC detectors - AND cond
    if cathode_indices and anodeV_indices and anodeH_indices:

        # Find the closest event in each detector (smallest absolute time differe
        cathode_diffs = np.abs(cathode_timetags[cathode_indices] - imp_timetag)
        anodeV_diffs = np.abs(anodeV_timetags[anodeV_indices] - imp_timetag)
        anodeH_diffs = np.abs(anodeH_timetags[anodeH_indices] - imp_timetag)

        closest_cathode_idx = cathode_indices[np.argmin(cathode_diffs)]
        closest_anodeV_idx = anodeV_indices[np.argmin(anodeV_diffs)]
        closest_anodeH_idx = anodeH_indices[np.argmin(anodeH_diffs)]

        # Get the corresponding rows
        cathode_data = cathode_sorted.iloc[closest_cathode_idx]
        anodeV_data = anodeV_sorted.iloc[closest_anodeV_idx]
        anodeH_data = anodeH_sorted.iloc[closest_anodeH_idx]

        # Calculate time difference values (in picoseconds)
        # +ve = PPAC after IMP, -ve = PPAC before IMP
        dt_cathode_ps = cathode_data['timetag'] - imp_timetag
        dt_anodeV_ps = anodeV_data['timetag'] - imp_timetag
```

```
            dt_anodeH_ps = anodeH_data['timetag'] - imp_timetag

            # Create event data dictionary with all relevant information
            event_data = {
                # IMP data
                'imp_timetag': imp_timetag,
                'imp_x': imp_row['x'],
                'imp_y': imp_row['y'],
                'imp_tagx': imp_row['tagx'],
                'imp_tagy': imp_row['tagy'],
                'imp_nfile': imp_row['nfile'],
                'imp_tdelta': imp_row['tdelta'],
                'imp_nX': imp_row['nX'],
                'imp_nY': imp_row['nY'],
                'imp_xE': imp_row['xE'],
                'imp_yE': imp_row['yE'],
                'xboard': imp_row['xboard'],
                'yboard': imp_row['yboard'],

                # Cathode data
                'cathode_timetag': cathode_data['timetag'],
                'cathode_energy': cathode_data['energy'],
                'cathode_board': cathode_data['board'],
                'cathode_channel': cathode_data['channel'],
                'cathode_nfile': cathode_data['nfile'],

                # AnodeV data
                'anodeV_timetag': anodeV_data['timetag'],
                'anodeV_energy': anodeV_data['energy'],
                'anodeV_board': anodeV_data['board'],
                'anodeV_channel': anodeV_data['channel'],
                'anodeV_nfile': anodeV_data['nfile'],

                # AnodeH data
                'anodeH_timetag': anodeH_data['timetag'],
                'anodeH_energy': anodeH_data['energy'],
                'anodeH_board': anodeH_data['board'],
                'anodeH_channel': anodeH_data['channel'],
                'anodeH_nfile': anodeH_data['nfile'],

                # Time difference values (in picoseconds)
                'dt_cathode_ps': dt_cathode_ps,
                'dt_anodeV_ps': dt_anodeV_ps,
                'dt_anodeH_ps': dt_anodeH_ps,

                # Convert to nanoseconds for convenience
                'dt_cathode_ns': dt_cathode_ps / 1000,
                'dt_anodeV_ns': dt_anodeV_ps / 1000,
                'dt_anodeH_ns': dt_anodeH_ps / 1000
            }

            coincident_events.append(event_data)
        else:
            non_coincident_data = {
                # IMP data
                'timetag': imp_timetag,
                't':imp_timetag / 1e12,
                'x': imp_row['x'],
                'y': imp_row['y'],
                'tagx': imp_row['tagx'],
                'tagy': imp_row['tagy'],
                'nfile': imp_row['nfile'],
                'tdelta': imp_row['tdelta'],
                'nX': imp_row['nX'],
                'nY': imp_row['nY'],
                'xE': imp_row['xE'],
                'yE': imp_row['yE'],
                'xboard': imp_row['xboard'],
                'yboard': imp_row['yboard'],
            }
```

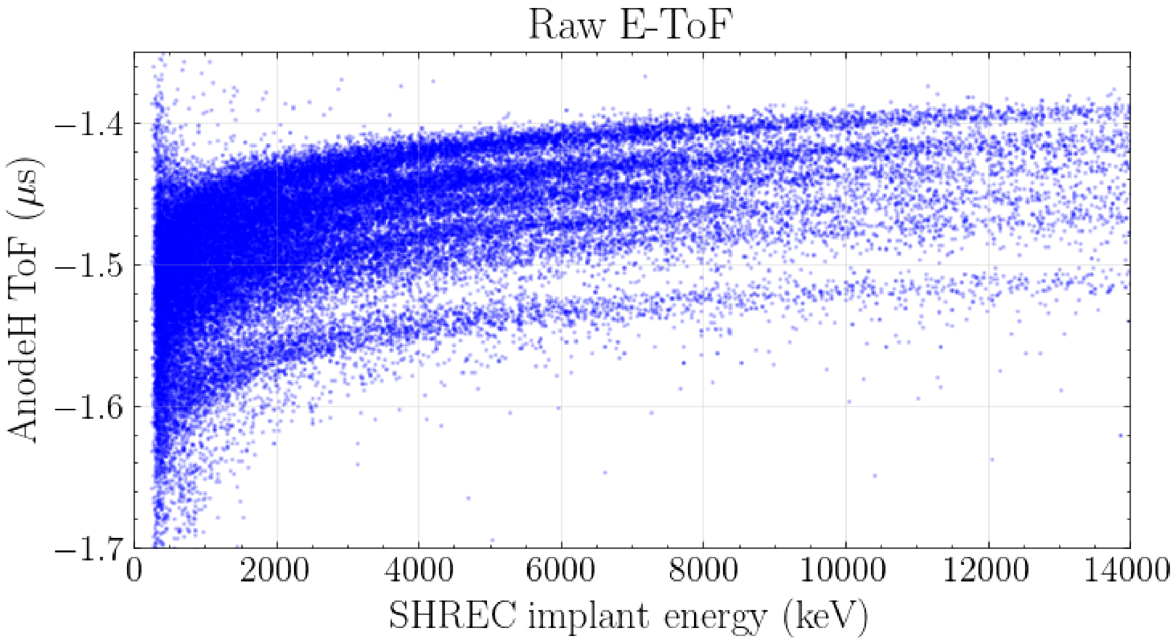
```
            non_ppac_coincident_events.append(non_coincident_data)

            # TODO - Since we use an AND condition between the ppac plates, we need t
            #        make sure theres no ppac signal at all.

        # Print progress every 10,000 events
        if idx % 10000 == 0 and idx > 0:
            elapsed = time.time() - start_time
            events_per_sec = idx / elapsed
            remaining_time = (total_imp_events - idx) / events_per_sec if events_per_
            print(f"Processed {idx}/{total_imp_events} events ({idx/total_imp_events*

# Create the df with coincident events
coincident_imp_df = pd.DataFrame(coincident_events)
non_coincident_imp_df = pd.DataFrame(non_ppac_coincident_events)
print(f"Found {len(coincident_imp_df)} coincidences within the window")

# Calculate total processing time
elapsed_time = time.time() - start_time
print(f"Total processing time: {elapsed_time:.2f} seconds")
print(f"Processing rate: {total_imp_events/elapsed_time:.1f} events/second")
```

```
Processing 693937 IMP events...
Processed 10000/693937 events (1.4%) - Rate: 9014.3 events/sec - ETA: 75.9 sec
Processed 20000/693937 events (2.9%) - Rate: 9714.1 events/sec - ETA: 69.4 sec
Processed 30000/693937 events (4.3%) - Rate: 10146.9 events/sec - ETA: 65.4 sec
Processed 40000/693937 events (5.8%) - Rate: 10419.8 events/sec - ETA: 62.8 sec
Processed 50000/693937 events (7.2%) - Rate: 10546.2 events/sec - ETA: 61.1 sec
Processed 60000/693937 events (8.6%) - Rate: 10656.5 events/sec - ETA: 59.5 sec
Processed 70000/693937 events (10.1%) - Rate: 10616.3 events/sec - ETA: 58.8 sec
Processed 80000/693937 events (11.5%) - Rate: 10567.7 events/sec - ETA: 58.1 sec
Processed 90000/693937 events (13.0%) - Rate: 10627.8 events/sec - ETA: 56.8 sec
Processed 100000/693937 events (14.4%) - Rate: 10622.3 events/sec - ETA: 55.9 se
c
Processed 110000/693937 events (15.9%) - Rate: 10606.9 events/sec - ETA: 55.1 se
c
Processed 120000/693937 events (17.3%) - Rate: 10599.0 events/sec - ETA: 54.2 se
c
Processed 130000/693937 events (18.7%) - Rate: 10571.3 events/sec - ETA: 53.3 se
c
Processed 140000/693937 events (20.2%) - Rate: 10613.4 events/sec - ETA: 52.2 se
c
Processed 150000/693937 events (21.6%) - Rate: 10638.8 events/sec - ETA: 51.1 se
c
Processed 160000/693937 events (23.1%) - Rate: 10569.4 events/sec - ETA: 50.5 se
c
Processed 170000/693937 events (24.5%) - Rate: 10498.7 events/sec - ETA: 49.9 se
c
Processed 180000/693937 events (25.9%) - Rate: 10469.5 events/sec - ETA: 49.1 se
c
Processed 190000/693937 events (27.4%) - Rate: 10448.8 events/sec - ETA: 48.2 se
c
Processed 200000/693937 events (28.8%) - Rate: 10425.2 events/sec - ETA: 47.4 se
c
Processed 210000/693937 events (30.3%) - Rate: 10381.0 events/sec - ETA: 46.6 se
c
Processed 220000/693937 events (31.7%) - Rate: 10365.6 events/sec - ETA: 45.7 se
c
Processed 230000/693937 events (33.1%) - Rate: 10380.2 events/sec - ETA: 44.7 se
c
Processed 240000/693937 events (34.6%) - Rate: 10391.6 events/sec - ETA: 43.7 se
c
Processed 250000/693937 events (36.0%) - Rate: 10382.0 events/sec - ETA: 42.8 se
c
Processed 260000/693937 events (37.5%) - Rate: 10344.8 events/sec - ETA: 41.9 se
c
Processed 270000/693937 events (38.9%) - Rate: 10306.4 events/sec - ETA: 41.1 se
c
Processed 280000/693937 events (40.3%) - Rate: 10309.7 events/sec - ETA: 40.2 se
c
Processed 290000/693937 events (41.8%) - Rate: 10329.5 events/sec - ETA: 39.1 se
c
Processed 300000/693937 events (43.2%) - Rate: 10344.2 events/sec - ETA: 38.1 se
c
Processed 310000/693937 events (44.7%) - Rate: 10363.9 events/sec - ETA: 37.0 se
c
Processed 320000/693937 events (46.1%) - Rate: 10364.7 events/sec - ETA: 36.1 se
c
Processed 330000/693937 events (47.6%) - Rate: 10379.2 events/sec - ETA: 35.1 se
c
Processed 340000/693937 events (49.0%) - Rate: 10407.3 events/sec - ETA: 34.0 se
c
Processed 350000/693937 events (50.4%) - Rate: 10434.5 events/sec - ETA: 33.0 se
c
Processed 360000/693937 events (51.9%) - Rate: 10449.9 events/sec - ETA: 32.0 se
c
Processed 370000/693937 events (53.3%) - Rate: 10471.7 events/sec - ETA: 30.9 se
c
Processed 380000/693937 events (54.8%) - Rate: 10494.7 events/sec - ETA: 29.9 se
c
Processed 390000/693937 events (56.2%) - Rate: 10508.0 events/sec - ETA: 28.9 se
c
```

```
Processed 400000/693937 events (57.6%) - Rate: 10527.2 events/sec - ETA: 27.9 se
c
Processed 410000/693937 events (59.1%) - Rate: 10548.5 events/sec - ETA: 26.9 se
c
Processed 420000/693937 events (60.5%) - Rate: 10564.5 events/sec - ETA: 25.9 se
c
Processed 430000/693937 events (62.0%) - Rate: 10581.7 events/sec - ETA: 24.9 se
c
Processed 440000/693937 events (63.4%) - Rate: 10595.8 events/sec - ETA: 24.0 se
c
Processed 450000/693937 events (64.8%) - Rate: 10610.5 events/sec - ETA: 23.0 se
c
Processed 460000/693937 events (66.3%) - Rate: 10591.3 events/sec - ETA: 22.1 se
c
Processed 470000/693937 events (67.7%) - Rate: 10567.0 events/sec - ETA: 21.2 se
c
Processed 480000/693937 events (69.2%) - Rate: 10561.1 events/sec - ETA: 20.3 se
c
Processed 490000/693937 events (70.6%) - Rate: 10570.4 events/sec - ETA: 19.3 se
c
Processed 500000/693937 events (72.1%) - Rate: 10583.2 events/sec - ETA: 18.3 se
c
Processed 510000/693937 events (73.5%) - Rate: 10595.5 events/sec - ETA: 17.4 se
c
Processed 520000/693937 events (74.9%) - Rate: 10607.1 events/sec - ETA: 16.4 se
c
Processed 530000/693937 events (76.4%) - Rate: 10620.3 events/sec - ETA: 15.4 se
c
Processed 540000/693937 events (77.8%) - Rate: 10633.2 events/sec - ETA: 14.5 se
c
Processed 550000/693937 events (79.3%) - Rate: 10647.5 events/sec - ETA: 13.5 se
c
Processed 560000/693937 events (80.7%) - Rate: 10656.1 events/sec - ETA: 12.6 se
c
Processed 570000/693937 events (82.1%) - Rate: 10651.0 events/sec - ETA: 11.6 se
c
Processed 580000/693937 events (83.6%) - Rate: 10660.5 events/sec - ETA: 10.7 se
c
Processed 590000/693937 events (85.0%) - Rate: 10667.5 events/sec - ETA: 9.7 sec
Processed 600000/693937 events (86.5%) - Rate: 10674.8 events/sec - ETA: 8.8 sec
Processed 610000/693937 events (87.9%) - Rate: 10680.5 events/sec - ETA: 7.9 sec
Processed 620000/693937 events (89.3%) - Rate: 10687.3 events/sec - ETA: 6.9 sec
Processed 630000/693937 events (90.8%) - Rate: 10693.3 events/sec - ETA: 6.0 sec
Processed 640000/693937 events (92.2%) - Rate: 10698.0 events/sec - ETA: 5.0 sec
Processed 650000/693937 events (93.7%) - Rate: 10689.8 events/sec - ETA: 4.1 sec
Processed 660000/693937 events (95.1%) - Rate: 10691.1 events/sec - ETA: 3.2 sec
Processed 670000/693937 events (96.6%) - Rate: 10681.1 events/sec - ETA: 2.2 sec
Processed 680000/693937 events (98.0%) - Rate: 10666.1 events/sec - ETA: 1.3 sec
Processed 690000/693937 events (99.4%) - Rate: 10652.0 events/sec - ETA: 0.4 sec
Found 90889 coincidences within the window
Total processing time: 68.29 seconds
Processing rate: 10161.6 events/second
```

Plot raw etof

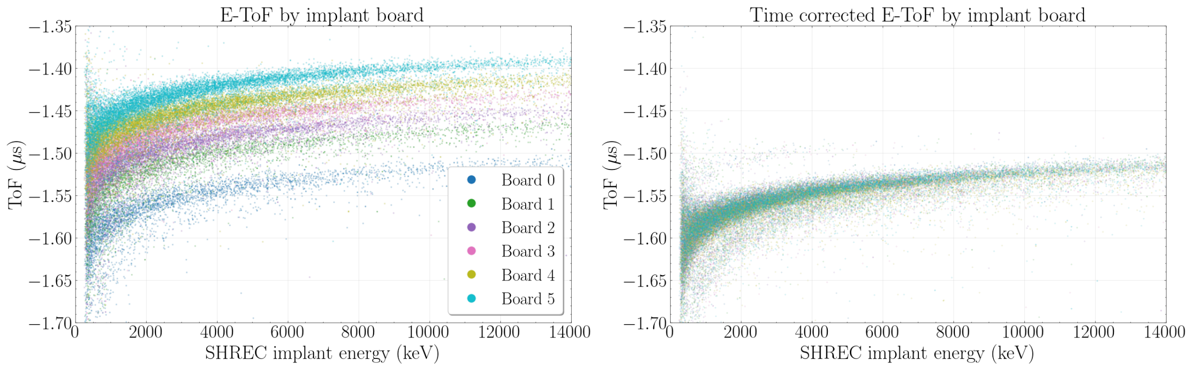
In [7]:
```
if not coincident_imp_df.empty:
    # Convert ns time differences to us for plotting
    coincident_imp_df['dt_cathode_us'] = coincident_imp_df['dt_cathode_ns'] / 100
    coincident_imp_df['dt_anodeV_us'] = coincident_imp_df['dt_anodeV_ns'] / 1000
    coincident_imp_df['dt_anodeH_us'] = coincident_imp_df['dt_anodeH_ns'] / 1000

    plt.figure(figsize=(8, 4))
    fs = 18
    plt.scatter(coincident_imp_df['imp_xE'], coincident_imp_df['dt_anodeH_us'],
                alpha=0.2, s=1, c='blue')
    plt.xlabel("SHREC implant energy (keV)", fontsize=fs)
    plt.ylabel(r"AnodeH ToF ($\mu$s)", fontsize=fs)
    plt.title("Raw E-ToF", fontsize=fs+2)
    plt.xlim(0, 14000)
    plt.ylim(-1.7, -1.35)
    plt.grid(True, alpha=0.3)
    ax = plt.gca()
    ax.tick_params(axis='both', which='major', labelsize=fs-2)
#     plt.legend(fontsize=fs-4, frameon=True)
    plt.savefig("plots/raw_etof.pdf", dpi=1000)
else:
    print("No coincidences")
```



Time correction for SHREC imp region boards

In [8]:
```
from matplotlib.lines import Line2D

# Get the recoil time in seconds
coincident_imp_df['t'] = coincident_imp_df['imp_timetag'] * 1e-12

# Define manual time offsets for the boards- board0 is master
manual_offsets = {
    0: 0,
    1: -0.045e-6,
    2: -0.065e-6,
    3: -0.085e-6,
    4: -0.105e-6,
    5: -0.125e-6,
}

# Calculate the corrected dt for the ppac plates in microseconds
# Staying consistent with xboard
coincident_imp_df['dt_anodeH_us_corr'] = coincident_imp_df.apply(
    lambda row: row['dt_anodeH_us'] + manual_offsets.get(row['xboard'], 0) * 1e6,
    axis=1
)

coincident_imp_df['dt_anodeV_us_corr'] = coincident_imp_df.apply(
    lambda row: row['dt_anodeV_us'] + manual_offsets.get(row['xboard'], 0) * 1e6,
    axis=1
)

coincident_imp_df['dt_cathode_us_corr'] = coincident_imp_df.apply(
    lambda row: row['dt_cathode_us'] + manual_offsets.get(row['xboard'], 0) * 1e6
    axis=1
)

# Get boards
boards = sorted(coincident_imp_df['xboard'].unique())

plt.figure(figsize=(30,18))
fs=30

plt.subplot(221)
colors = plt.cm.tab10(np.linspace(0, 1, len(boards)))
legend_handles = []

for board, color in zip(boards, colors):
    # Filter the df for this board
    board_data = coincident_imp_df[coincident_imp_df['xboard'] == board]
    plt.scatter(board_data['imp_xE'], board_data['dt_anodeH_us'],
                s=2, alpha=0.2, color=color, label=f'Board {board}')
    legend_handles.append(Line2D([0], [0], marker='o', color='w', markersize=14,
plt.xlabel("SHREC implant energy (keV)", fontsize=fs)
plt.ylabel(r"ToF ($\mu$s)", fontsize=fs)
plt.title("E-ToF by implant board", fontsize=fs+2)
plt.xlim(0, 14000)
plt.ylim(-1.7, -1.35)
plt.grid(True, alpha=0.3)
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)
plt.legend(handles=legend_handles, fontsize=fs-4, frameon=True, shadow=True, face

plt.subplot(222)
for board, color in zip(boards, colors):
    # Filter the DataFrame for this board
    board_data = coincident_imp_df[coincident_imp_df['xboard'] == board]
    plt.scatter(board_data['imp_xE'], board_data['dt_anodeH_us_corr'],
                s=2, alpha=0.1, color=color, label=f'Board {board}')
    legend_handles.append(Line2D([0], [0], marker='o', color='w', markersize=14,
plt.xlabel("SHREC implant energy (keV)", fontsize=fs)
plt.ylabel(r"ToF ($\mu$s)", fontsize=fs)
plt.title("Time corrected E-ToF by implant board", fontsize=fs+2)
plt.xlim(0, 14000)
plt.ylim(-1.7, -1.35)
```

```
plt.grid(True, alpha=0.3)
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)
# plt.legend(handles=legend_handles, fontsize=fs-4, frameon=True)


plt.savefig("plots/etof_by_board.png", dpi=1000)
```



Decay events

In [9]:
```
# Set decay time window
min_corr_time = 0.00000001     # Minimum time after recoil to consider (in second
max_corr_time = 20     # Maximum time after recoil to consider (in seconds)
```

In [10]:
```
# Build pixel history from the imp df & group the full implant event history by p
pixel_groups = imp.groupby(['x', 'y'])
pixel_history = {pixel: group for pixel, group in pixel_groups}
```

In [11]:
```
# Create decay event list
decay_events = []
```

In [12]:
```
# For each recoil event, search for subsequent events in the same pixel from imp

# Create decay events list to hold events
decay_candidates = []

# Loop through coincident imp (recoil-like) events
for recoil_idx, recoil in coincident_imp_df.iterrows():

    # Get the pixel for the recoil event
    pixel = (recoil['imp_x'], recoil['imp_y'])

    # Convert the recoil imp_timetag from picoseconds to seconds
    recoil_time_sec = recoil['imp_timetag'] / 1e12

    # Check if there are any events in the same pixel in the imp region df.
    if pixel not in pixel_history:
        continue  # Skip if no events are found for this pixel

    # Get the time sorted events for this pixel from imp
    pixel_df = pixel_history[pixel]

    # Get the pixel time values as a sorted array
    time_array = pixel_df['t'].values # This is in seconds

    # Define the lower and upper bounds for candidate decay events in seconds
    lower_bound = recoil_time_sec + min_corr_time
    upper_bound = recoil_time_sec + max_corr_time

    # Use binary search to find the index positions in the time array
    start_idx = np.searchsorted(time_array, lower_bound, side='left')
    end_idx = np.searchsorted(time_array, upper_bound, side='right')

    # If events exist in the correlation window, add them as candidate decay even
    if start_idx < end_idx:

        candidate_events = pixel_df.iloc[start_idx:end_idx].copy()

        # Record the associated recoil info for later
        candidate_events['recoil_index'] = recoil_idx
        candidate_events['recoil_time_sec'] = recoil_time_sec
        decay_candidates.append(candidate_events) # add decay candidates to list

# Combine all candidate decay events into a single df
if decay_candidates:
    decay_candidates_df = pd.concat(decay_candidates, ignore_index=True)
else:
    decay_candidates_df = pd.DataFrame()

# Display the first few decay candidates
print(decay_candidates_df.head())
```

```
          t    x   y           tagx           tagy  nfile  xboard  yboard  \
0  0.198635    9  50   198634810650   198634760742      0       4       7
1  0.529692  147  12   529691550542   529691391264      0       0       7
2  2.839951  147  12  2839951104007  2839951009995      0       0       7
3  0.912401   18  11   912400550245   912400538374      0       5       6
4  0.916485   40  19   916484660872   916484657870      0       5       6

   tdelta  nX  nY            xE            yE event_type  recoil_index  \
0   49908   1   1   6782.193936   6774.234035        imp             0
1  159278   1   1  38852.690963  38278.948495        imp             1
2   94012   1   1    333.187114    416.077084        imp             1
3   11871   1   1   2687.152713   2689.662740        imp             2
4    3002   1   1   2320.956539   2315.324385        imp             3

   recoil_time_sec
0         0.198635
1         0.529692
2         0.529692
3         0.912401
4         0.916485
```

PPAC Anticoincidence check for decays

Check the candidate decay is in the non-coincident list, do this by merging on pixel?

In [13]:

```
# Check the unique (x, y, t) keys in each DataFrame
print("Decay candidates unique keys:", decay_candidates_df[['x', 'y', 't']].drop_
print("Non-coincident unique keys:", non_coincident_imp_df[['x', 'y', 't']].drop_
```

```
Decay candidates unique keys: (56421, 3)
Non-coincident unique keys: (603048, 3)
```

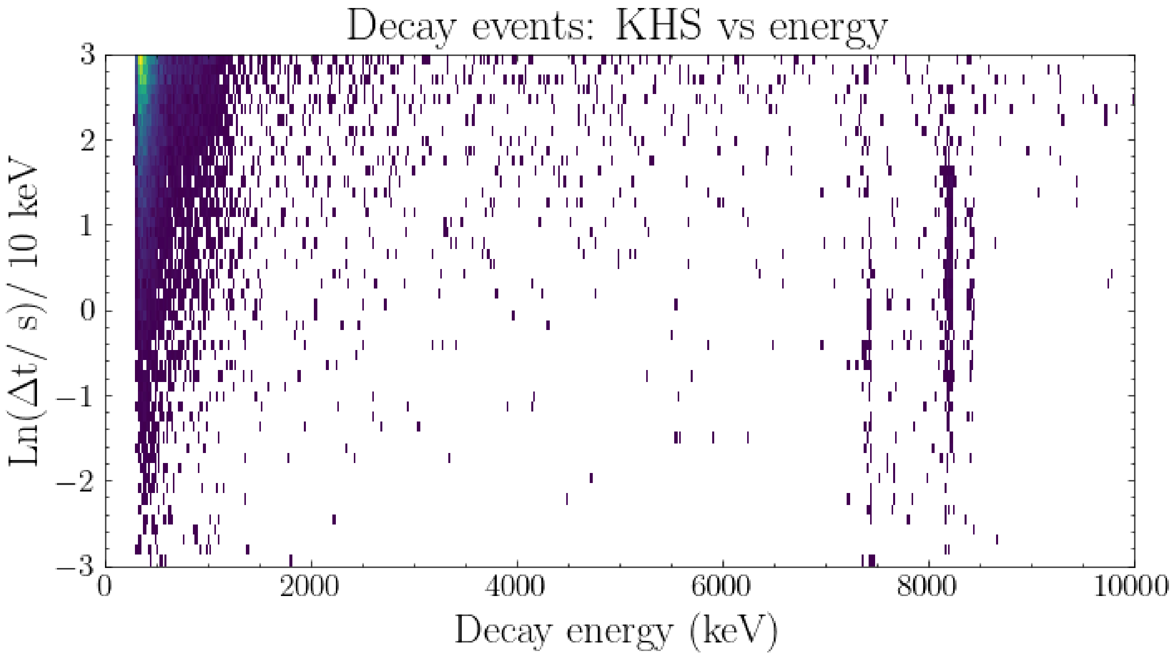
In [14]:
```
if not decay_candidates_df.empty:
    # Drop duplicate rows based on x and y in non_coincident_imp_df
    non_coincident_clean = non_coincident_imp_df[['x', 'y']].drop_duplicates()

    # every row in decay_candidates_df is kept,
    # and we add data from non_coincident_clean where there is a match on x and y
    decay_candidates_df = decay_candidates_df.merge(
        non_coincident_clean,
        on=['x', 'y'],
        how='left',
        indicator='ppac_flag'
    )

    # If an event from decay_candidates_df finds a matching row in non_coincident
    # ppac_flag will be set to "both".
    # If there is no match (i.e. PPAC signal), ppac_flag will be 'left_only'
    decay_candidates_df['is_clean'] = decay_candidates_df['ppac_flag'] == 'both'

print(decay_candidates_df['is_clean'].value_counts())
print(decay_candidates_df.head())
```

```
is_clean
True    57312
Name: count, dtype: int64
          t    x   y           tagx           tagy  nfile  xboard  yboard  \
0  0.198635    9  50   198634810650   198634760742      0       4       7
1  0.529692  147  12   529691550542   529691391264      0       0       7
2  2.839951  147  12  2839951104007  2839951009995      0       0       7
3  0.912401   18  11   912400550245   912400538374      0       5       6
4  0.916485   40  19   916484660872   916484657870      0       5       6

   tdelta  nX  nY            xE            yE event_type  recoil_index  \
0   49908   1   1   6782.193936   6774.234035        imp             0
1  159278   1   1  38852.690963  38278.948495        imp             1
2   94012   1   1    333.187114    416.077084        imp             1
3   11871   1   1   2687.152713   2689.662740        imp             2
4    3002   1   1   2320.956539   2315.324385        imp             3

   recoil_time_sec ppac_flag  is_clean
0         0.198635      both      True
1         0.529692      both      True
2         0.529692      both      True
3         0.912401      both      True
4         0.916485      both      True
```

Decay KHS

In [15]:
```
# Find the log time between implant and decay event
decay_candidates_df['log_dt'] = np.log(abs(decay_candidates_df['t'] - decay_candi
```

In [16]:
```
# Plot the 2d KHS hist
fs = 18
plt.figure(figsize=(8,4))
plt.hist2d(decay_candidates_df['yE'], decay_candidates_df['log_dt'],
           bins=((500),(50)), range=((0,10000),(-3,3)), cmin=1)
plt.xlabel('Decay energy (keV)', fontsize=fs)
plt.ylabel(r'Ln($\Delta$t/ s)/ 10 keV', fontsize=fs)
plt.title('Decay events: KHS vs energy', fontsize=fs+2)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=fs-4)
plt.savefig('plots/decay_khs.pdf', dpi=1000)
```



EVR-a correlations

In [17]:
```
# Alpha energy, time gates
# Recoil energy gates

alpha_energy_min = 8150    # Minimum alpha energy (keV)
alpha_energy_max = 8300    # Maximum alpha energy (keV)

recoil_energy_min = 2000   # Minimum recoil energy (keV)
recoil_energy_max = 8099  # Maximum recoil energy (keV)

alpha_corr_min = 0.08     # Minimum time difference in seconds
alpha_corr_max = 15         # Maximum time difference in seconds
```

In [18]:
```
# Filter alpha candidates by energy
filtered_alpha_candidates = decay_candidates_df[
    (decay_candidates_df['xE'] >= alpha_energy_min) &
    (decay_candidates_df['xE'] <= alpha_energy_max)
].copy()
```

In [19]:
```
# just making sure we have t
if 't' not in filtered_alpha_candidates.columns:
    filtered_alpha_candidates['t'] = filtered_alpha_candidates['timetag'] / 1e12
```

In [20]:
```
# for each alpha candidate, find the preceeding recoil in same pixel

# initialising cols in the df
filtered_alpha_candidates['closest_recoil_index'] = np.nan
filtered_alpha_candidates['recoil_time'] = np.nan
filtered_alpha_candidates['time_difference'] = np.nan
filtered_alpha_candidates['recoil_energy'] = np.nan

# loop through the alpha candidates
for idx, alpha in filtered_alpha_candidates.iterrows():
    pixel_x = alpha['x']
    pixel_y = alpha['y']
    alpha_time = alpha['t']

    # Retrieve all recoil events from the same pixel
    recoils_in_pixel = coincident_imp_df[
        (coincident_imp_df['imp_x'] == pixel_x) & (coincident_imp_df['imp_y'] ==
    ]

    # apply recoil energy gate
    recoils_in_pixel = recoils_in_pixel[
        (recoils_in_pixel['imp_xE'] >= recoil_energy_min) &
        (recoils_in_pixel['imp_xE'] <= recoil_energy_max)
    ]

    # Only consider recoils that occurred before the alpha event
    recoils_before = recoils_in_pixel[recoils_in_pixel['t'] < alpha_time]

    if not recoils_before.empty:

        # its good to work with copies... compute the time difference between r-a
        recoils_before = recoils_before.copy()
        recoils_before['time_diff'] = alpha_time - recoils_before['t']

        # make sure the r-a fits in the coincidence window
        recoils_in_window = recoils_before[
            (recoils_before['time_diff'] >= alpha_corr_min) &
            (recoils_before['time_diff'] <= alpha_corr_max)
        ]

        if not recoils_in_window.empty:
            # there might be multiple correlations, so choose the one with the sm
            closest_recoil = recoils_in_window.loc[recoils_in_window['time_diff']
            filtered_alpha_candidates.at[idx, 'closest_recoil_index'] = closest_r
            filtered_alpha_candidates.at[idx, 'recoil_time'] = closest_recoil['t'
            filtered_alpha_candidates.at[idx, 'time_difference'] = closest_recoil
            filtered_alpha_candidates.at[idx, 'recoil_energy'] = closest_recoil['
        else:
            filtered_alpha_candidates.at[idx, 'closest_recoil_index'] = np.nan
            filtered_alpha_candidates.at[idx, 'recoil_time'] = np.nan
            filtered_alpha_candidates.at[idx, 'time_difference'] = np.nan
            filtered_alpha_candidates.at[idx, 'recoil_energy'] = np.nan
    else:
        filtered_alpha_candidates.at[idx, 'closest_recoil_index'] = np.nan
        filtered_alpha_candidates.at[idx, 'recoil_time'] = np.nan
        filtered_alpha_candidates.at[idx, 'time_difference'] = np.nan
        filtered_alpha_candidates.at[idx, 'recoil_energy'] = np.nan
```

In [21]:
```
# Build the correlation dataframe
correlated_events = filtered_alpha_candidates.dropna(subset=['recoil_time']).copy
print("Number of correlated alpha-recoil events:", len(correlated_events))
print(correlated_events.head())
```

```
Number of correlated alpha-recoil events: 345
              t   x   y             tagx             tagy  nfile  xboard  \
367   63.474081  35  15   63474081082902   63474081040154      0       4   
556   97.735318  87  26   97735318457340   97735318370716      0       2   
585  101.742032  15   9  101742032315555  101742032286685      0       4   
869  146.771431  68  10  146771431178029  146771431113087      1       3   
871  146.771431  68   9  146771431178029  146771431125371      1       3   

     yboard  tdelta  nX  ...  event_type  recoil_index  recoil_time_sec  \
367       6   42748   2  ...         imp           556        60.462240   
556       7   86624   1  ...         imp           846        93.449282   
585       6   28870   1  ...         imp           892        99.922332   
869       7   64942   2  ...         imp          1322       145.821629   
871       6   52658   2  ...         imp          1323       145.821629   

    ppac_flag  is_clean    log_dt closest_recoil_index  recoil_time  \
367      both      True  1.102552                556.0    60.462240   
556      both      True  1.455362                846.0    93.449282   
585      both      True  0.598671                892.0    99.922332   
869      both      True -0.051502               1322.0   145.821629   
871      both      True -0.051502               1323.0   145.821629   

     time_difference  recoil_energy  
367         3.011841    6862.952065  
556         4.286036    4388.934390  
585         1.819700    6385.057232  
869         0.949802    5916.498072  
871         0.949802    5916.498072  

[5 rows x 23 columns]
```

In [22]:
```
# Merge the recoil and alpha info together, and rename things for clarity
recoil_rename = {
    'imp_timetag': 'rec_timetag',
    'imp_x': 'rec_x',
    'imp_y': 'rec_y',
    'imp_tagx': 'rec_tagx',
    'imp_tagy': 'rec_tagy',
    'imp_nfile': 'rec_nfile',
    'imp_tdelta': 'rec_tdelta',
    'imp_nX': 'rec_nX',
    'imp_nY': 'rec_nY',
    'imp_xE': 'rec_xE',
    'imp_yE': 'rec_yE',
    'xboard': 'rec_xboard',
    'yboard': 'rec_yboard',
    'cathode_timetag': 'rec_cathode_timetag',
    'cathode_energy': 'rec_cathode_energy',
    'cathode_board': 'rec_cathode_board',
    'cathode_channel': 'rec_cathode_channel',
    'cathode_nfile': 'rec_cathode_nfile',
    'anodeV_timetag': 'rec_anodeV_timetag',
    'anodeV_energy': 'rec_anodeV_energy',
    'anodeV_board': 'rec_anodeV_board',
    'anodeV_channel': 'rec_anodeV_channel',
    'anodeV_nfile': 'rec_anodeV_nfile',
    'anodeH_timetag': 'rec_anodeH_timetag',
    'anodeH_energy': 'rec_anodeH_energy',
    'anodeH_board': 'rec_anodeH_board',
    'anodeH_channel': 'rec_anodeH_channel',
    'anodeH_nfile': 'rec_anodeH_nfile',
    'dt_cathode_ps': 'rec_dt_cathode_ps',
    'dt_anodeV_ps': 'rec_dt_anodeV_ps',
    'dt_anodeH_ps': 'rec_dt_anodeH_ps',
    'dt_cathode_ns': 'rec_dt_cathode_ns',
    'dt_anodeV_ns': 'rec_dt_anodeV_ns',
    'dt_anodeH_ns': 'rec_dt_anodeH_ns',
    'dt_cathode_us': 'rec_dt_cathode_us',
    'dt_anodeV_us': 'rec_dt_anodeV_us',
    'dt_anodeH_us': 'rec_dt_anodeH_us',
    't': 'rec_t',
    'dt_anodeH_us_corr': 'rec_dt_anodeH_us_corr',
    'dt_anodeV_us_corr': 'rec_dt_anodeV_us_corr',
    'dt_cathode_us_corr': 'rec_dt_cathode_us_corr'
}

alpha_rename = {
    't': 'alpha_t',
    'x': 'alpha_x',
    'y': 'alpha_y',
    'tagx': 'alpha_tagx',
    'tagy': 'alpha_tagy',
    'nfile': 'alpha_nfile',
    'xboard': 'alpha_xboard',
    'yboard': 'alpha_yboard',
    'tdelta': 'alpha_tdelta',
    'nX': 'alpha_nX',
    'nY': 'alpha_nY',
    'xE': 'alpha_xE',
    'yE': 'alpha_yE',
    'event_type': 'alpha_event_type',
    'recoil_index': 'alpha_recoil_index',
    'recoil_time_sec': 'alpha_recoil_time',
    'ppac_flag': 'alpha_ppac_flag',
    'is_clean': 'alpha_is_clean',
    'log_dt': 'alpha_log_dt',
    # Also include new computed cols
    'closest_recoil_index': 'alpha_closest_recoil_index',
    'recoil_time': 'alpha_recoil_time_calculated',
    'time_difference': 'alpha_time_difference',
    'recoil_energy': 'alpha_recoil_energy'
```

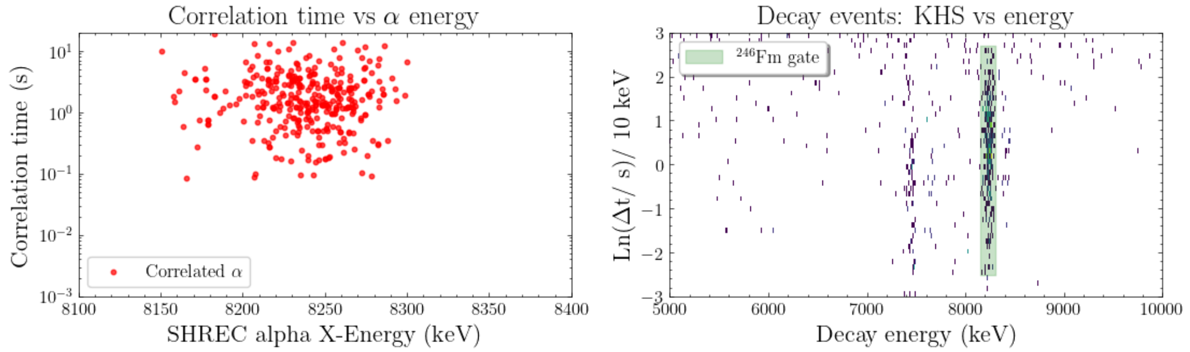
```
}

# Rename columns in the recoil df
recoil_df_renamed = coincident_imp_df.copy().rename(columns=recoil_rename)

# Rename columns in the alpha df
alpha_df_renamed = correlated_events.copy().rename(columns=alpha_rename)

# Merge the two dfs using the recoil index
final_correlated_df = alpha_df_renamed.merge(
    recoil_df_renamed,
    left_on='alpha_recoil_index',
    right_index=True,
    how='left'
)
```

In [23]:
```
# print some check stuff
print("Final correlated Events df:")
print(final_correlated_df.head())
print("Checking pixel matches (alpha vs. recoil):")
print(final_correlated_df[['alpha_x', 'alpha_y', 'rec_x', 'rec_y']].head())
```

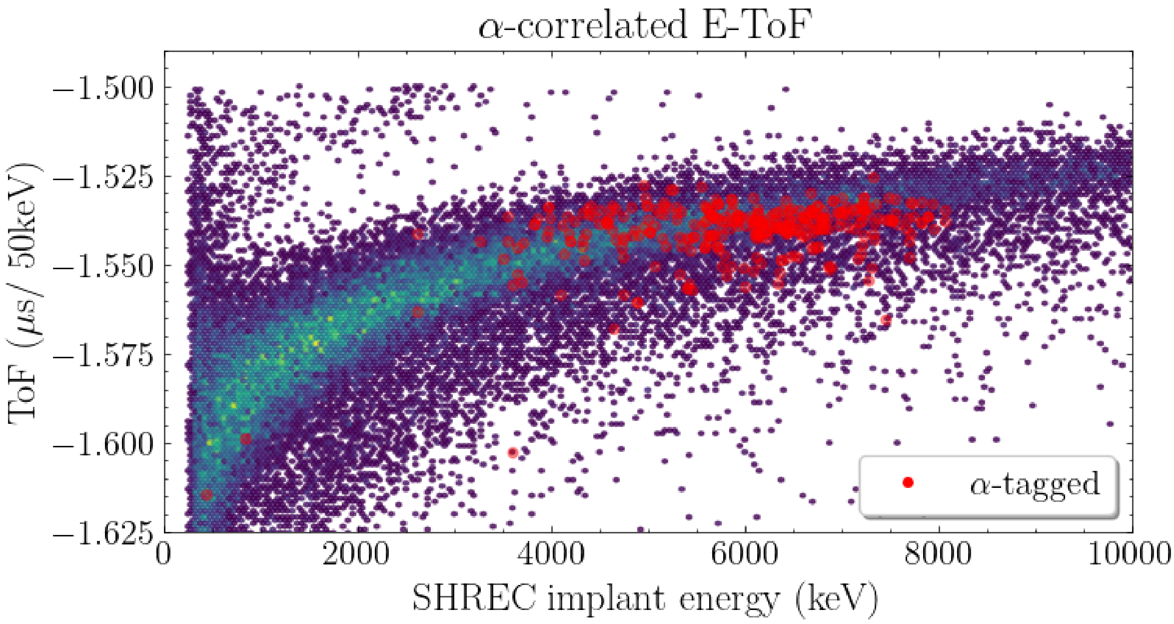
```
Final correlated Events df:
        alpha_t  alpha_x  alpha_y       alpha_tagx       alpha_tagy  \
367   63.474081       35       15   63474081082902   63474081040154
556   97.735318       87       26   97735318457340   97735318370716
585  101.742032       15        9  101742032315555  101742032286685
869  146.771431       68       10  146771431178029  146771431113087
871  146.771431       68        9  146771431178029  146771431125371

     alpha_nfile  alpha_xboard  alpha_yboard  alpha_tdelta  alpha_nX  ...  \
367            0             4             6         42748         2  ...
556            0             2             7         86624         1  ...
585            0             4             6         28870         1  ...
869            1             3             7         64942         2  ...
871            1             3             6         52658         2  ...

     rec_dt_cathode_ns  rec_dt_anodeV_ns  rec_dt_anodeH_ns rec_dt_cathode_us  \
367          -1439.042         -1444.590         -1445.423         -1.439042
556          -1462.898         -1469.603         -1471.085         -1.462898
585          -1426.733         -1433.706         -1434.065         -1.426733
869          -1445.170         -1452.925         -1450.838         -1.445170
871          -1445.170         -1452.925         -1450.838         -1.445170

     rec_dt_anodeV_us  rec_dt_anodeH_us       rec_t  rec_dt_anodeH_us_corr  \
367         -1.444590         -1.445423   60.462240              -1.550423
556         -1.469603         -1.471085   93.449282              -1.536085
585         -1.433706         -1.434065   99.922332              -1.539065
869         -1.452925         -1.450838  145.821629              -1.535838
871         -1.452925         -1.450838  145.821629              -1.535838

     rec_dt_anodeV_us_corr  rec_dt_cathode_us_corr
367              -1.549590               -1.544042
556              -1.534603               -1.527898
585              -1.538706               -1.531733
869              -1.537925               -1.530170
871              -1.537925               -1.530170

[5 rows x 64 columns]
Checking pixel matches (alpha vs. recoil):
     alpha_x  alpha_y  rec_x  rec_y
367       35       15     35     15
556       87       26     87     26
585       15        9     15      9
869       68       10     68     10
871       68        9     68      9
```

Plotting correlated stuff

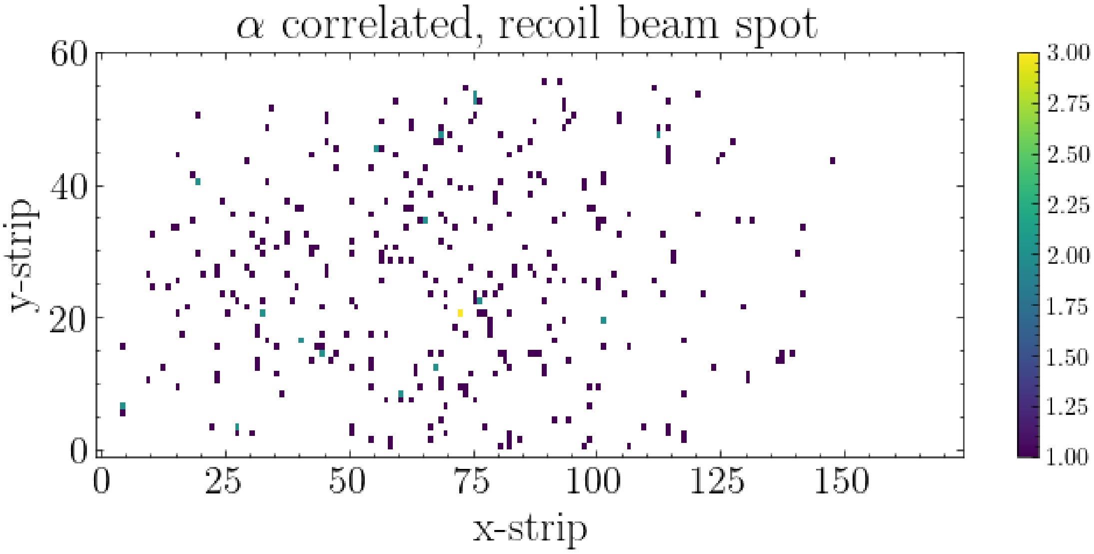
In [24]:
```
# log decay time

final_correlated_df['log_dt'] = np.log10(np.abs(final_correlated_df['alpha_t'] -
final_correlated_df['rec_alpha_time'] =np.abs(final_correlated_df['alpha_t'] - fi
fs = 16
plt.figure(figsize=(13,7))

plt.subplot(221)
plt.scatter(final_correlated_df['alpha_xE'], final_correlated_df['rec_alpha_time'
            s=10, color='red', alpha=0.7, label=r'Correlated $\alpha$')
plt.xlabel('SHREC alpha X-Energy (keV)', fontsize=fs)
# plt.ylabel(r'log(dt/s)', fontsize=fs)
plt.ylabel(r'Correlation time (s)', fontsize=fs)
plt.xlim(8100, 8400)
plt.yscale('log')
ax = plt.gca()
ax.tick_params(axis='both', labelsize=fs-4 )
plt.legend(fontsize=fs-4, loc='lower left', frameon=True)
plt.ylim(0.001,20)
plt.title(r'Correlation time vs $\alpha$ energy', fontsize=fs+2)

plt.subplot(222)
plt.hist2d(decay_candidates_df['xE'], decay_candidates_df['log_dt'],
           bins=((500),(50)), range=((5000,10000),(-3,3)), cmin=1)
plt.fill_betweenx(y=[np.log(alpha_corr_min), np.log(alpha_corr_max)], x1=alpha_en
                  color='g', alpha=0.2, label=r'$^{246}$Fm gate')
plt.xlabel('Decay energy (keV)', fontsize=fs)
plt.ylabel(r'Ln($\Delta$t/ s)/ 10 keV', fontsize=fs)
plt.title('Decay events: KHS vs energy', fontsize=fs+2)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=fs-4)
plt.legend(fontsize=fs-4, loc='upper left', frameon=True, facecolor='white', shad




plt.savefig('plots/log_time_corr_alphas.pdf', dpi=300)
```



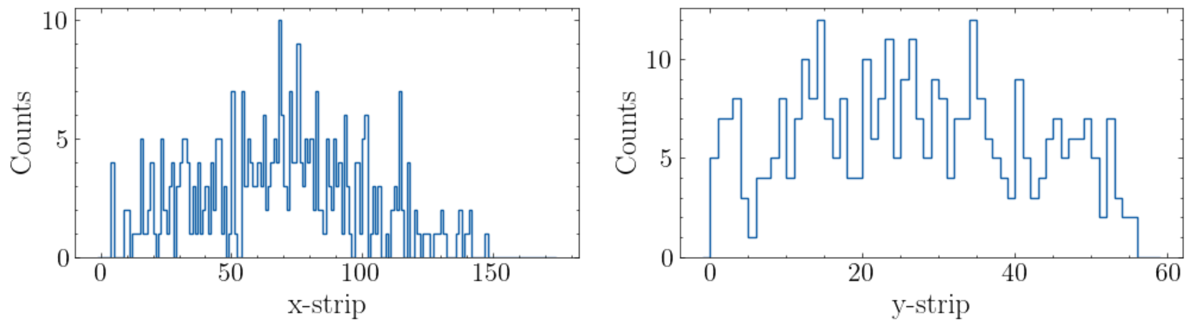
In [25]:
```
# Correlated etof

plt.figure(figsize=(8,4))
fs = 18
plt.hexbin(coincident_imp_df['imp_xE'], coincident_imp_df['dt_anodeH_us_corr'],
           gridsize=200, extent=(0, 10000, -1.7, -1.5), mincnt=1, cmap='viridis',
plt.scatter(final_correlated_df['rec_xE'], final_correlated_df['rec_dt_anodeH_us_
            color='red', alpha=0.4, s=20, label=r'$\alpha$-tagged')
legend_marker = Line2D([0], [0], marker='o', color='w', markersize=6,
                       markerfacecolor='red', label=r'$\alpha$-tagged')
plt.ylim(-1.625, -1.49)
plt.xlim(0, 10000)
plt.xlabel('SHREC implant energy (keV)', fontsize=fs)
plt.ylabel(r'ToF ($\mu$s/ 50keV)', fontsize=fs)
plt.title(r'$\alpha$-correlated E-ToF', fontsize=fs+2)
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)
plt.legend(handles=[legend_marker], loc='lower right', fontsize=fs-2, frameon=Tru
plt.savefig('plots/correlated_etof.pdf', dpi=300)
```



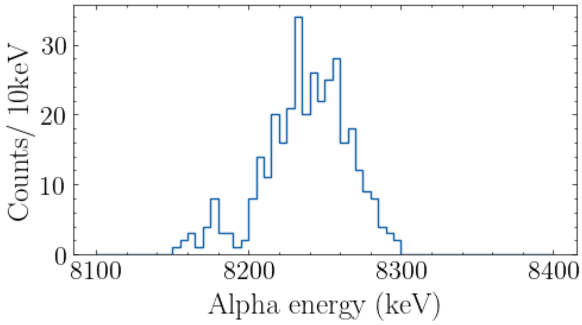
In [26]:
```
# correlated beam spot
plt.figure(figsize=(8,3))
fs = 18
plt.hist2d(final_correlated_df['rec_x'], final_correlated_df['rec_y'],
           bins=((175),(61)), range=((-1,174),(-1,60)), cmin=1)
# plt.xlim(0, 10000)
plt.xlabel('x-strip', fontsize=fs)
plt.ylabel(r'y-strip', fontsize=fs)
plt.title(r'$\alpha$ correlated, recoil beam spot', fontsize=fs+2)
plt.colorbar()
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)
# plt.legend(loc='lower right', fontsize=fs-2, frameon=True)
plt.savefig('plots/correlated_stripX_stripY.pdf', dpi=300)
```

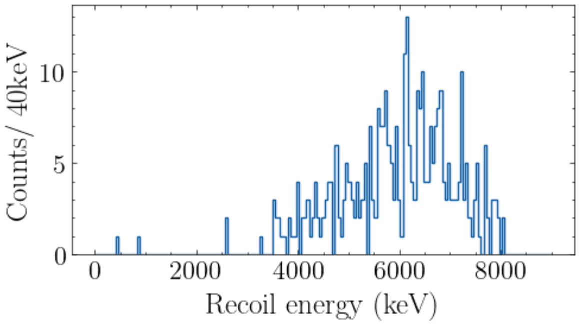


In [27]:
```
# beam spot projections
# correlated beam spot

plt.figure(figsize=(12,6))
fs = 18

plt.subplot(221)
plt.hist(final_correlated_df['rec_x'], histtype='step', bins=175, range=(-1,174))
plt.xlabel('x-strip', fontsize=fs)
plt.ylabel(r'Counts', fontsize=fs)
# plt.title(r'$\alpha$ correlated, recoil beam spot', fontsize=fs+2)
# plt.colorbar()
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)

plt.subplot(222)
plt.hist(final_correlated_df['rec_y'], histtype='step',bins=60, range=(-1,59))
plt.xlabel('y-strip', fontsize=fs)
plt.ylabel(r'Counts', fontsize=fs)
# plt.title(r'$\alpha$ correlated, recoil beam spot', fontsize=fs+2)
# plt.colorbar()
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)


plt.savefig('plots/correlated_beam_spot_projections.pdf', dpi=300)
```



In [31]:
```
# Recoil and alpha energies

plt.figure(figsize=(12,6))
fs = 18

plt.subplot(221)
plt.hist(final_correlated_df['alpha_xE'], histtype='step', bins=60, range=(8100,8
plt.xlabel('Alpha energy (keV)', fontsize=fs)
plt.ylabel(r'Counts/ 10keV', fontsize=fs)
# plt.title(r'$\alpha$ correlated, recoil beam spot', fontsize=fs+2)
# plt.colorbar()
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)

plt.subplot(222)
plt.hist(final_correlated_df['rec_xE'], histtype='step',bins=175, range=(0,9000))
plt.xlabel('Recoil energy (keV)', fontsize=fs)
plt.ylabel(r'Counts/ 40keV', fontsize=fs)
ax = plt.gca()
ax.tick_params(axis='both', which='major', labelsize=fs-2)

plt.savefig('plots/rec_alpha_energy_projections.pdf', dpi=300)
```





In [ ]:

In [ ]: