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/dssd_non_vetoed_events.csv') # n
ppac = pd.read_csv('processed_data/ppac_events.csv') # raw, uncalib
ruth = pd.read_csv('processed_data/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  # 2000 ns (2 us) before
window_after_ns = 5000   # 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=Tr
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)  # Usin
```

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 t
```

In [6]:
```
# Function to find PPAC events within the time window
def find_events_in_window(imp_timetag, detector_timetags, window_be
    """
    Find ppac events that occur within the specified time window ar
    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 picoseco
    window_after_ps : Time window after the IMP event in picosecond

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

    # 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

    # Find ppac events in time window
    cathode_indices = find_events_in_window(imp_timetag, cathode_ti
    anodeV_indices = find_events_in_window(imp_timetag, anodeV_time
    anodeH_indices = find_events_in_window(imp_timetag, anodeH_time

    # Only proceed if we have coincidences in all three PPAC detect
    if cathode_indices and anodeV_indices and anodeH_indices:

        # Find the closest event in each detector (smallest absolut
        cathode_diffs = np.abs(cathode_timetags[cathode_indices] -
        anodeV_diffs = np.abs(anodeV_timetags[anodeV_indices] - imp
        anodeH_diffs = np.abs(anodeH_timetags[anodeH_indices] - imp

        closest_cathode_idx = cathode_indices[np.argmin(cathode_dif
        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 informatio
        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)
```

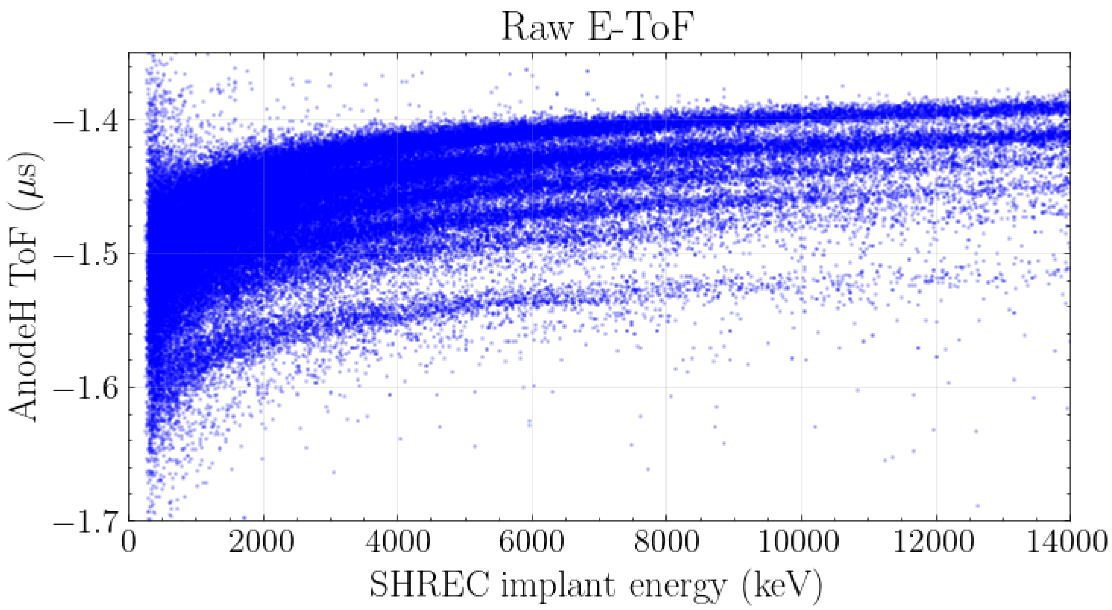
```
    else:
        non_coincident_data = {
            # IMP data
            'timetag': imp_timetag,
            't':imp_timetag / 1e12,
            'x': imp_row['x'],
            'y': imp_row['y'],
            'tagx': imp_row['tagx'],
            'tagy': imp_row['tagy'],
            'nfile': imp_row['nfile'],
            '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 pla
        #        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
        print(f"Processed {idx}/{total_imp_events} events ({idx/tot

# 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 wind

# 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
```

```
Processing 513316 IMP events...
Processed 10000/513316 events (1.9%) - Rate: 8106.7 events/sec - E
TA: 62.1 sec
Processed 20000/513316 events (3.9%) - Rate: 8551.7 events/sec - E
TA: 57.7 sec
Processed 30000/513316 events (5.8%) - Rate: 8800.2 events/sec - E
TA: 54.9 sec
Processed 40000/513316 events (7.8%) - Rate: 8970.2 events/sec - E
TA: 52.8 sec
Processed 50000/513316 events (9.7%) - Rate: 9077.1 events/sec - E
TA: 51.0 sec
Processed 60000/513316 events (11.7%) - Rate: 9148.0 events/sec -
ETA: 49.6 sec
Processed 70000/513316 events (13.6%) - Rate: 9193.1 events/sec -
ETA: 48.2 sec
Processed 80000/513316 events (15.6%) - Rate: 9227.3 events/sec -
ETA: 47.0 sec
Processed 90000/513316 events (17.5%) - Rate: 9265.0 events/sec -
ETA: 45.7 sec
Processed 100000/513316 events (19.5%) - Rate: 9289.7 events/sec -
ETA: 44.5 sec
Processed 110000/513316 events (21.4%) - Rate: 9077.3 events/sec -
ETA: 44.4 sec
Processed 120000/513316 events (23.4%) - Rate: 8949.0 events/sec -
ETA: 44.0 sec
Processed 130000/513316 events (25.3%) - Rate: 8889.8 events/sec -
ETA: 43.1 sec
Processed 140000/513316 events (27.3%) - Rate: 8676.6 events/sec -
ETA: 43.0 sec
Processed 150000/513316 events (29.2%) - Rate: 8526.8 events/sec -
ETA: 42.6 sec
Processed 160000/513316 events (31.2%) - Rate: 8428.1 events/sec -
ETA: 41.9 sec
Processed 170000/513316 events (33.1%) - Rate: 8365.8 events/sec -
ETA: 41.0 sec
Processed 180000/513316 events (35.1%) - Rate: 8239.9 events/sec -
ETA: 40.5 sec
Processed 190000/513316 events (37.0%) - Rate: 8285.1 events/sec -
ETA: 39.0 sec
Processed 200000/513316 events (39.0%) - Rate: 8202.8 events/sec -
ETA: 38.2 sec
Processed 210000/513316 events (40.9%) - Rate: 8196.9 events/sec -
ETA: 37.0 sec
Processed 220000/513316 events (42.9%) - Rate: 8249.0 events/sec -
ETA: 35.6 sec
Processed 230000/513316 events (44.8%) - Rate: 8294.8 events/sec -
ETA: 34.2 sec
Processed 240000/513316 events (46.8%) - Rate: 8335.7 events/sec -
ETA: 32.8 sec
Processed 250000/513316 events (48.7%) - Rate: 8381.1 events/sec -
ETA: 31.4 sec
Processed 260000/513316 events (50.7%) - Rate: 8419.1 events/sec -
ETA: 30.1 sec
Processed 270000/513316 events (52.6%) - Rate: 8457.7 events/sec -
ETA: 28.8 sec
Processed 280000/513316 events (54.5%) - Rate: 8476.2 events/sec -
ETA: 27.5 sec
Processed 290000/513316 events (56.5%) - Rate: 8510.9 events/sec -
ETA: 26.2 sec
Processed 300000/513316 events (58.4%) - Rate: 8544.0 events/sec -
ETA: 25.0 sec
```

```
Processed 310000/513316 events (60.4%) - Rate: 8551.0 events/sec -
ETA: 23.8 sec
Processed 320000/513316 events (62.3%) - Rate: 8515.0 events/sec -
ETA: 22.7 sec
Processed 330000/513316 events (64.3%) - Rate: 8434.7 events/sec -
ETA: 21.7 sec
Processed 340000/513316 events (66.2%) - Rate: 8459.7 events/sec -
ETA: 20.5 sec
Processed 350000/513316 events (68.2%) - Rate: 8484.5 events/sec -
ETA: 19.2 sec
Processed 360000/513316 events (70.1%) - Rate: 8475.0 events/sec -
ETA: 18.1 sec
Processed 370000/513316 events (72.1%) - Rate: 8467.2 events/sec -
ETA: 16.9 sec
Processed 380000/513316 events (74.0%) - Rate: 8488.5 events/sec -
ETA: 15.7 sec
Processed 390000/513316 events (76.0%) - Rate: 8509.7 events/sec -
ETA: 14.5 sec
Processed 400000/513316 events (77.9%) - Rate: 8533.8 events/sec -
ETA: 13.3 sec
Processed 410000/513316 events (79.9%) - Rate: 8553.8 events/sec -
ETA: 12.1 sec
Processed 420000/513316 events (81.8%) - Rate: 8578.1 events/sec -
ETA: 10.9 sec
Processed 430000/513316 events (83.8%) - Rate: 8601.1 events/sec -
ETA: 9.7 sec
Processed 440000/513316 events (85.7%) - Rate: 8620.0 events/sec -
ETA: 8.5 sec
Processed 450000/513316 events (87.7%) - Rate: 8613.6 events/sec -
ETA: 7.4 sec
Processed 460000/513316 events (89.6%) - Rate: 8630.7 events/sec -
ETA: 6.2 sec
Processed 470000/513316 events (91.6%) - Rate: 8652.4 events/sec -
ETA: 5.0 sec
Processed 480000/513316 events (93.5%) - Rate: 8670.6 events/sec -
ETA: 3.8 sec
Processed 490000/513316 events (95.5%) - Rate: 8688.0 events/sec -
ETA: 2.7 sec
Processed 500000/513316 events (97.4%) - Rate: 8704.9 events/sec -
ETA: 1.5 sec
Processed 510000/513316 events (99.4%) - Rate: 8719.7 events/sec -
ETA: 0.4 sec
Found 111067 coincidences within the window
Total processing time: 61.25 seconds
Processing rate: 8380.8 events/second
```

Plot raw etof

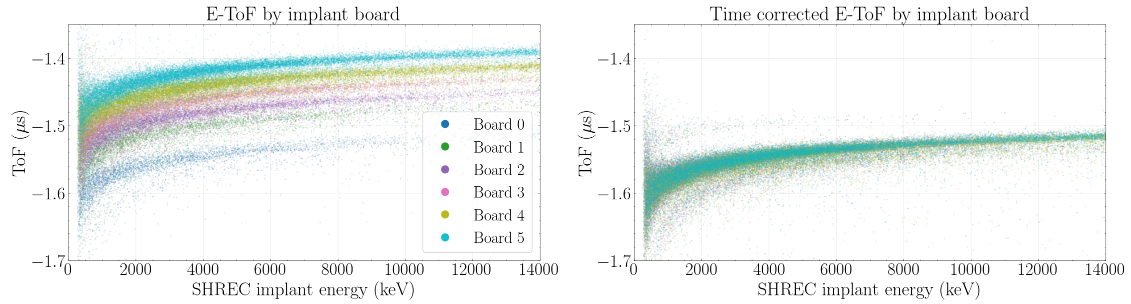
In [7]:
```
if not coincident_imp_df.empty:
    # Convert ns time differences to us for plotting
    coincident_imp_df['dt_cathode_us'] = coincident_imp_df['dt_cath
    coincident_imp_df['dt_anodeV_us'] = coincident_imp_df['dt_anode
    coincident_imp_df['dt_anodeH_us'] = coincident_imp_df['dt_anode

    plt.figure(figsize=(8, 4))
    fs = 18
    plt.scatter(coincident_imp_df['imp_xE'], coincident_imp_df['dt_
                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['xboar
    axis=1
)

coincident_imp_df['dt_anodeV_us_corr'] = coincident_imp_df.apply(
    lambda row: row['dt_anodeV_us'] + manual_offsets.get(row['xboar
    axis=1
)

coincident_imp_df['dt_cathode_us_corr'] = coincident_imp_df.apply(
    lambda row: row['dt_cathode_us'] + manual_offsets.get(row['xboa
    axis=1
)

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

plt.figure(figsize=(30,15))
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'] == b
    plt.scatter(board_data['imp_xE'], board_data['dt_anodeH_us'],
                s=2, alpha=0.1, color=color, label=f'Board {board}'
    legend_handles.append(Line2D([0], [0], marker='o', color='w', m
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)

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'] == b
    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', m
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 = 0.0001     # Minimum time after recoil to consider
max_corr_time = 10     # Maximum time after recoil to consider (in
```

In [10]:
```
# Build pixel history from the imp df & group the full implant even
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 p

# 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 re
    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
    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 arr
    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 candid
    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 candi

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

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

```
            t   x   y           tagx           tagy  nfile  xboar
d  yboard  \
0   8.687956  44  49   8687955632004   8687955664003      0
5       6
1   5.053563  18   6   5053563376005   5053563200001      0
5       7
2   6.452244   1  12   6452244162993   6452244115993      0
4       7
3  15.641202  92  14  15641202472000  15641202341994      0
3       7
4   7.341375   5   6   7341374909494   7341374867187      0
4       7

   tdelta  nX  nY          xE          yE event_type  recoil_ind
ex  \
0  -31999   1   1   334.541983   355.716979        imp
52
1  176004   1   1   289.114464   359.805537        imp
55
2   47000   1   2   865.410811   574.482571        imp
61
3  130006   1   1   303.709971   356.048970        imp
78
4   42307   1   1  3865.706690  3916.650417        imp
81

   recoil_time_sec
0         4.073180
1         4.598866
2         4.716802
3         6.079251
4         6.369336
```

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
print("Non-coincident unique keys:", non_coincident_imp_df[['x', 'y
```

```
Decay candidates unique keys: (9845, 3)
Non-coincident unique keys: (402249, 3)
```

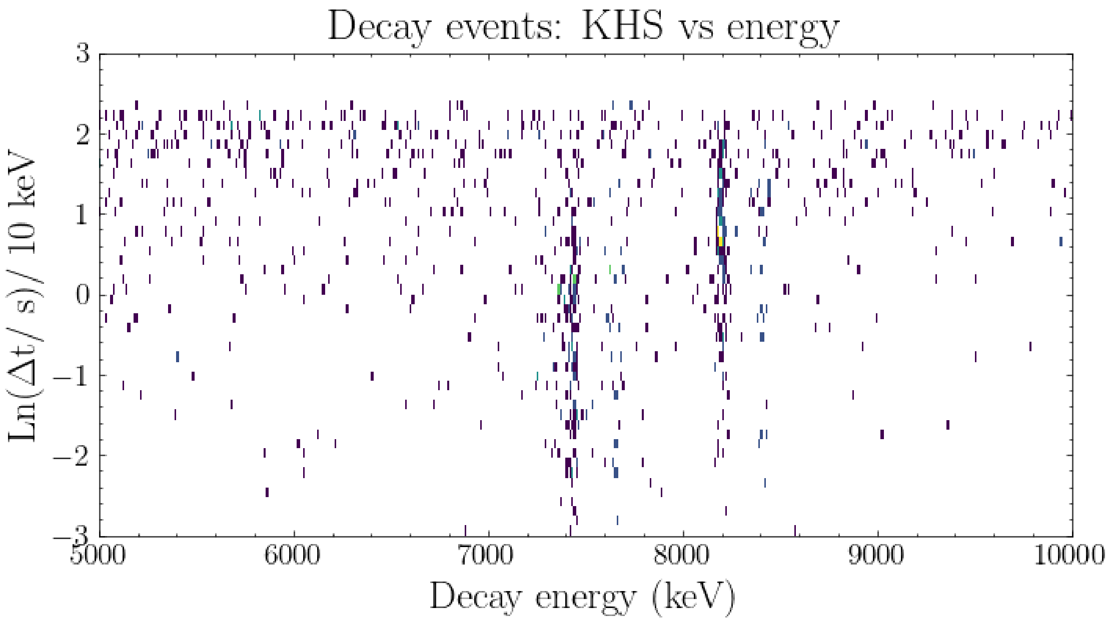
In [14]:
```
if not decay_candidates_df.empty:
    # Drop duplicate rows based on x and y in non_coincident_imp_df
    non_coincident_clean = non_coincident_imp_df[['x', 'y']].drop_d

    # every row in decay_candidates_df is kept,
    # and we add data from non_coincident_clean where there is a ma
    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
    # ppac_flag will be set to "both".
    # If there is no match (i.e. PPAC signal), ppac_flag will be 'l
    decay_candidates_df['is_clean'] = decay_candidates_df['ppac_fla

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

```
is_clean
True    9986
Name: count, dtype: int64
           t   x   y           tagx           tagy  nfile  xboar
d  yboard  \
0   8.687956  44  49   8687955632004   8687955664003      0
5       6
1   5.053563  18   6   5053563376005   5053563200001      0
5       7
2   6.452244   1  12   6452244162993   6452244115993      0
4       7
3  15.641202  92  14  15641202472000  15641202341994      0
3       7
4   7.341375   5   6   7341374909494   7341374867187      0
4       7

   tdelta  nX  nY          xE          yE event_type  recoil_ind
ex  \
0  -31999   1   1   334.541983   355.716979        imp
52
1  176004   1   1   289.114464   359.805537        imp
55
2   47000   1   2   865.410811   574.482571        imp
61
3  130006   1   1   303.709971   356.048970        imp
78
4   42307   1   1  3865.706690  3916.650417        imp
81

   recoil_time_sec ppac_flag  is_clean
0         4.073180      both      True
1         4.598866      both      True
2         4.716802      both      True
3         6.079251      both      True
4         6.369336      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']
```

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=((5000,10000),(-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)
```



Tighter gates on recoils and decays for EVR-a correlations

In [17]:
```
decay_candidates_df.head()
```

Out[17]:

	t	x	y	tagx	tagy	nfile	xboard	yboard	tdelta	nX
0	8.687956	44	49	8687955632004	8687955664003	0	5	6	-31999	1
1	5.053563	18	6	5053563376005	5053563200001	0	5	7	176004	1
2	6.452244	1	12	6452244162993	6452244115993	0	4	7	47000	1
3	15.641202	92	14	15641202472000	15641202341994	0	3	7	130006	1
4	7.341375	5	6	7341374909494	7341374867187	0	4	7	42307	1